MASTER EN SCIENCES INFORMATIQUES

Enseignement assisté par ordinateur : "l'équilibre chimique"

Bertrand, J.L.

Award date:
1986

Awarding institution:
Universite de Namur

Année Académique 1985-1986

ENSEIGNEMENT ASSISTE PAR ORDINATEUR : "L'EQUILIBRE CHIMIQUE"

Promoteur : Mr C. CHERTON
Mémoire présenté par JL BERTRAND
en vue de l'obtention du grade
de licencié et maître
en informatique

Je tiens à exprimer mes remerciements à tous ceux qui, d'une manière ou d'une autre, ont contribué à l'élaboration de cet ouvrage. Plus particulièrement à Monsieur CHERTON, promoteur de ce mémoire, pour les nombreux encouragements et judicieux conseils qu'il m'a prodigués tout au long de ce travail ainsi qu'à Monsieur PIRSON, professeur de chimie, pour les fructueuses et encourageantes discussions que nous avons eues au cours de cette année.

TABLES DES MATIERES

Introduction : L'enseignement assisté par ordinateur et son utilisation 1

Préliminaires 4

1. Qualités d'un didactitiel 4
2. Quel type d'enseignement assisté par ordinateur ? 5
   - 2.1. Enseignement assisté par ordinateur directif 5
   - 2.2. Enseignement assisté par ordinateur non directif 7
3. L'équilibre chimique : deux approches didactiques 9
4. En quoi l'ordinateur peut-il nous aider ? 11
5. Choix de la solution à développer ... 13
6. Présentation générale du travail..... 14

Chapitre I : Approche déductive..... 15

- 1.1. Description du modèle 15
- 1.2. Simulation d'une réaction chimique 22

Chapitre II: Approche inductive 24

- 2.1. Exemple 24
- 2.2. Généralisation 39

Chapitre III : Analyse fonctionnelle 44
3.1. Présentation du langage 44
3.2. Obtention des concentrations à l'équilibre 47
3.3. Calcul de la valeur de la constante d'équilibre 49
3.4. Schéma global du système 52
3.4.1. Définition des traitements 55
3.4.2. Définition des données. 56

Chapitre IV : Analyse organique 59
4.1. Découpe en modules 59
4.2. Justification de la découpe en modules 63

Chapitre V : Illustration de la réutilisation des modules 65
5.1. Variation de la constante d'équilibre en fonction de la température 65
5.2. Influence de la température sur la réversibilité des réactions. 67

Conclusion 71

Bibliographiques 71

Annexes 76

INTRODUCTION

L'enseignement assisté par ordinateur et son utilisation

Les responsables de l'enseignement ont à faire face à un double problème posé chaque année en des termes plus aigüs :

- la croissance constante de la population scolaire;
- l'augmentation de la quantité d'information à transmettre.

D'autre part, les réflexions pédagogiques sur les objectifs de l'enseignement ont mis en évidence l'importance de la formation en regard de l'information. Les systèmes d'enseignement assisté par ordinateur tentent d'apporter une solution à la fois à ces difficultés et exigences. En effet, contrairement aux moyens audio-visuels, ils permettent le plus souvent une individualisation suffisante de l'apprentissage. Toutefois, il faut rester conscient que l'enseignement assisté par ordinateur ne peut, à lui seul, satisfaire à tous les buts de l'enseignement. Il enrichit l'enseignement et lui procure de puissantes techniques; ces techniques sont cependant complémentaires. Les ordinateurs ne constituent en rien une solution miracle aux problèmes de formation scolaire. Ils ne sont vraiment efficaces que lorsqu'ils s'intègrent dans une perspective qui leur donne leur véritable fonction : celle d'un moyen parmi d'autres pour atteindre des

buts. Si ces buts ne sont pas définis, l'enseignement assisté par ordinateur se révèlera tôt ou tard plus nuisible qu'utile. L'enseignement assisté par ordinateur ne se conçoit que comme un des éléments d'une technologie de l'enseignement dans la perspective plus globale d'un système d'enseignement au service d'une politique d'enseignement.

Vers une communication informaticiens- enseignants

Profitant de la baisse de prix du matériel et de l'introduction du cours d'informatique au programme des cours de l'enseignement secondaire, l'informatique, en tant qu'aide à l'enseignement, a connu ces dernières années un développement considérable. Le prix des micro-ordinateurs ne cessant de diminuer, ils seront bientôt accessibles à toutes les écoles et ce malgré les restrictions imposées actuellement par le gouvernement.

Profitant de leur introduction dans l'enseignement, de nombreux programmes ont été développés par des firmes commerciales. Pour séduire le client, celles-ci ont établi des didacticiels (ensemble de programmes à objectif pédagogique) qui vantent leur technologie attrayante et une ergonomie poussée. Cependant, ces programmes, agréables à utiliser, sont généralement dotés d'une valeur pédagogique très discutable. Le peu d'impact pédagogique de ces logiciels peut se justifier par une faible participation, voire une abscence totale de parti-

cipation des enseignants. Aussi, pour que des innovations pédagogiques aient des chances de s'imposer, il faut absolument que les enseignants soient impliqués dans leur élaboration. Un bon programme d'enseignement assisté par ordinateur doit être bon non seulement sur le plan informatique mais également sur le plan didactique.

PRELIMINAIRES

1 Qualités d'un didactitiel

En plus de sa valeur pédagogique, le didactitiel développé devra jouir d'autres propriétés. Ainsi, il pourra facilement être :

1.1 Modifié

Le logiciel pourra être adapté aux désirs et besoins de l'enseignant sinon celui-ci ne l'acceptera pas comme tel. En effet, ce dernier a l'habitude d'utiliser des exercices qui lui sont personnels pour illustrer la matière à enseigner.

1.2 Porté

La plupart des logiciels consacrés à l'enseignement assisté par ordinateur sont destinés à être implémentés sur micro-ordinateurs. Vu la multitude de micro-ordinateurs actuellement disponibles sur le marché et l'absence de standards en la matière, il est indispensable de limiter et de localiser au maximum les dépendances vis à vis d'un matériel particulier.

1.3 Réutilisé

Si les enseignants sont désireux de programmer eux-mêmes leurs propres applications, il serait utile que ceux-ci disposent de composants qui soient réutilisables

dans d'autres contextes. La fonction de chaque composant devra être spécifiée de la façon la plus précise possible.

1.4 Extensible

Compte tenu du fait que de plus en plus d'enseignants reçoivent une formation en informatique, ceux-ci seront peut être désireux d'étendre le didactitiel mis à leur disposition.

1.5 Documenté

Les logiciels devront être accompagnés de toute la documentation nécessaire à leur compréhension. Celle-ci permettra de modifier ou d'étendre facilement le système.

2 Quel type d'enseignement assisté par ordinateur ?

On distingue généralement deux formes d'enseignement assisté par ordinateur : une forme directive et une forme non directive. Nous allons brièvement en donner les principales caractéristiques ainsi que leurs avantages et inconvénients respectifs.

2.1 Enseignement assisté par ordinateur directif

Celui-ci nécessite, pour un exercice particulier, la mémorisation de l'énoncé de la solution et éventuellement les erreurs courantes ainsi que les explications à fournir à l'élève. Le programme résultant est généralement assez simple à réaliser. Il consiste principalement en l'affichage de pages de scénario en fonction de la réponse donnée

par l'élève. On n'a pas besoin d'un apprentissage particulier pour utiliser le système. L'utilisateur déduit directement de la question posée, la réponse attendue. Les techniques employées ne sont pas particulières à la matière choisie. Le programme développé pourra donc être facilement adapté aux désirs et aux besoins de l'enseignant. Toutefois, celui-ci devra être initié à la programmation.

Cette forme d'enseignement assisté par ordinateur possède plusieurs avantages :

- ceux de l'enseignement programmé :
    * une participation active de l'élève;
    * un ordre efficace de présentation des concepts;
    * une correction immédiate et point par point de l'acquis;
    * une adaptation permanente aux difficultés d'assimilation de l'élève si le scénario est bien conçu.

- l'utilisation de l'ordinateur est attrayante et souvent synonyme de jeu;

- le programme est simple à réaliser.

Elle possède aussi certains inconvénients :

- ceux de l'enseignement programmé :
    * un certain ennui provoqué par le caractère monotone des séquences d'apprentissage;
    * un type d'enseignement impersonnel dû à l'absence du professeur.

- beaucoup de temps pour rédiger le scénario;

- beaucoup de temps pour ajouter un nouvel exercice;

- difficultés d'étendre le programme pour les enseignants non initiés à la programmation.

2.2 Enseignement assisté par ordinateur non directif

Il serait sûrement plus intéressant d'envisager une forme d'enseignement assisté par ordinateur où l'on dote la machine des "connaissances" nécessaires à l'enseignement de la matière désirée. Il serait ainsi possible de tirer avantage des capacités de calcul de l'ordinateur.

Cette forme d'enseignement assisté par ordinateur n'est cependant envisageable que si la matière est bien limitée et nécessite des réponses précises. En effet, il est exclu que l'on introduise dans un micro-ordinateur un ensemble de "connaissances" relatives à une branche très vaste. Les réponses doivent être précises pour que le programme qui les analysera ne soit pas trop complexe et reste donc implémentable sur un petit système.

L'idéal serait que, sur base d'un énoncé introduit par l'utilisateur, le programme soit capable d'en calculer la solution en donnant d'éventuelles explications sur la marche à suivre. Pour ce faire, l'énoncé devra être compréhensible à la fois par l'élève et par l'ordinateur. On utilisera donc un vocabulaire limité et une syntaxe c'est- à-dire un langage.

Contrairement aux questions/réponses ou menus à choix multiples de l'enseignement du premier type, ce langage permettra d'établir un vrai dialogue entre l'ordinateur et l'élève. Il sera donc possible de décrire le problème au moyen d'un langage familier à l'utilisateur.

Les techniques employées sont plus complexes mais peuvent être concentrées dans des modules de bas niveau. Ceux-ci devront être spécifiés de la façon la plus précise possible. L'apprentissage est peut être plus compliqué car on doit connaître la syntaxe du langage pour pouvoir introduire l'énoncé du problème. Cependant, le professeur pourra définir à tout instant la séquence d'apprentissage la mieux appropriée à l'élève. En effet, il pourra introduire lui_même, et en un minimum de temps, les exercices les mieux adaptés aux difficultés de celui-ci.

Cette forme d'enseignement assisté par ordinateur possède plusieurs <u>avantages</u> :

- ceux de l'enseignement programmé;
- l'utilisation attrayante de l'ordinateur;
- mieux adaptée aux difficultés d'assimilation de l'élève;
- l'ajout d'un exercice requiert un minimum de temps;
- communication plus riche entre l'élève et l'ordinateur.

Elle possède également quelques <u>inconvénients</u> :

- la programmation et les techniques utilisées sont plus complexes;

- les explications générées par le programme peuvent être moins explicites;

- limitée à une matière pas trop vaste et demandant des réponses précises aux questions posées.

3 L'équilibre chimique : deux approches didactiques

Ce travail d'enseignement assisté par ordinateur traitera de l'équilibre chimique. Pour introduire cette matière, deux types de démarche peuvent être envisagés.

Premièrement, une démarche inductive où l'on progresse des faits aux lois. Cette démarche consiste donc à partir des faits tirés de l'expérimentation pour essayer d'en tirer une loi aussi générale que possible. Cette démarche suppose donc que l'observation est le point de départ de la connaissance. Il faut cependant préciser qu'aucune connaissance inductive ne peut être considérée comme certaine : il est en effet rare que l'inventaire puisse être exhaustif et les observations n'étant pas répetables à l'infini, il est toujours à craindre qu'une nouvelle observation vienne démentir les précédentes. De plus, il ne faut pas oublier que chacune des expériences réalisées est entachée d'une erreur expérimentale.

Deuxièmement, une démarche déductive où l'on procède des principes à leurs applications et aux expériences qui les vérifient. Il peut être efficace et commode dans certains cas de déduire de principes généraux suggérés par l'expérience, des conséquences dont on

contrôlera à postériori l'accord avec les faits.

Cette démarche passe par l'élaboration d'un modèle ou d'une théorie. Ceux-ci permettront de décrire, c'est-à-dire de reproduire ,de façon aussi précise que possible les données qui ont permis de les construire. Ils permettront également de prévoir des résultats d'expérience.

Un modèle se présente donc comme un instrument permettant de relier les phénomènes et d'en prévoir d'autres. Un modèle possède à la fois une capacité descriptive et une capacité prévisionnelle. Toutefois, il est nécessaire de préciser qu'un modèle n'est jamais qu'une façon de décrire la réalité indépendammentde cette même réalité et que deux ou plusieurs modèles sont capables de rendre compte d'un même ensemble de phénomènes. Un modèle repose sur un certain nombre d'hypothèses et il devra être rejeté à partir du moment où il ne sera plus en accord avec les faits. En effet, il pourra toujours se trouver une ou plusieurs données expérimentales ne vérifiant pas le modèle. Celui-ci n'est donc pas une vérité établie une fois pour toute mais est sujet à des réactualisations.

Si le premier souci des scientifiques est d'obtenir des modèles universels, il est bien souvent apparu à postériori qu'un modèle ne décrit la réalité que dans un domaine bien particulier. Un modèle ne pourra donc être utilisé que dans la mesure où ses hypothèses d'applicabilité sont entièrement satisfaites.

Nous pensons qu'une démarche efficace et raison-

nable nécessite que l'on procède à la fois à une démarche inductive et à une démarche déductive.

Ce qui importe avant tout, c'est d'apprendre aux élèves à raisonner selon une méthode rigoureusement scientifique, de les familiariser avec une certaine façon de penser. Or la science est un moyen par lequel la connaissance s'améliore, non par des spéculations théoriques ou par une accumulation désordonnée d'observations, mais plutôt par une répétition raisonnée entre théorie et pratique. Ces deux approches sont donc complémentaires. En procédant de la sorte, les élèves ne seraient peut être plus amenés à utiliser des formules sans maîtriser leur champ d'application. Leur travail serait plus profond et irait plus dans le sens d'une compréhension réelle des mécanismes de résolution des problèmes. Peut-être éviterait-on de la sorte le recours à la mémorisation automatique des concepts par les élèves.

4 En quoi l'ordinateur peut-il nous aider ?

Devant une matière aussi fondamentale par le nombre d'applications qu'elle peut engendrer, il est indispensable que ce soit le professeur lui-même qui l'introduise. En effet, celui-ci est plus apte à mettre l'accent sur les caractéristiques de l'état d'équilibre et les conséquences de la réversibilité des réactions.

Toutefois, l'utilisation de l'ordinateur dans une telle matière peut se révèler extrêmement intéressante. En effet, l'ordinateur est un outil qui va permettre de rendre plus vécu et plus familier le concept d'équilibre

chimique.

D'une part, on peut développer un programme qui simule une réaction équilibrée afin de mieux se rendre compte de l'aspect dynamique de l'équilibre. L'objectif de ce modèle n'est pas de décrire comment une réaction a lieu, par quels intermédiaires elle passe, mais de visualiser graphiquement la réversibilité des réactions chimiques.

Rappelons quand même qu'une simulation ne vaut que par la précision de son modèle : elle représente la réalité de manière plus ou moins fidèle ou plus ou moins détaillée. Si le système réel est complexe, la simulation permet de dégrossir le problème en étudiant de façon indépendante les paramètres importants et l'incidence de leur variation. En particulier, la possibilité de faire varier les paramètres est importante pour une meilleure compréhension du modèle.

D'autre part, l'organisation des travaux pratiques en chimie se heurte à de grandes difficultés :

- coût excessif des appareils de laboratoire et de certains produits;

- danger de certaines expériences;

- temps insuffisant pour réaliser une expérience.

L'ordinateur apparaît encore ici comme un outil très intéressant dans la mesure où il permet de pallier à de telles difficultés. Avec lui, certaines expériences de laboratoire peuvent être simulées pour permettre d'obtenir directement les résultats de ces expériences.

A partir de ces résultats, il ne reste plus qu'à développer un programme qui aidera l'élève à retrouver la loi de GULDBERG-WAAGE. Soulignons que ces expériences "artificielles" ne remplacent jamais le contact direct avec l'expérience de laboratoire. Ici encore, l'ordinateur n'est qu'un outil de plus à la disposition du pédagogue.

5 Choix de la solution à développer

La seconde méthode nous paraît plus avantageuse car elle permet à l'utilisateur d'introduire directement et en un minimum de temps un nouvel exercice. Cette facilité est d'autant plus considérable que les enseignants ont souvent pris l'habitude d'illustrer leurs cours avec des exercices personnels. On aboutit ainsi à une certaine "personnalisation" du système. Les enseignants se sentiront peut être plus concernés par le didactitiel car celui-ci répond mieux à leurs désirs et besoins.
De plus, la manière d'introduire l'énoncé nous paraît pédagogiquement plus intéressante. En effet, en utilisant le langage adopté par les chimistes pour décrire une réaction chimique, un dialogue familier est établi entre l'élève et l'ordinateur. De plus, c'est l'occasion de vérifier si les élèves sont capables d'écrire correctement une équation chimique.

6 Présentation générale du travail

Le premier chapitre tentera de mettre au point un modèle de réaction chimique. Ce modèle permettrait de retrouver la loi de GULDBERG-WAAGE de façon déductive. Un programme de simulation pourrait être développé sur base de ce modèle. Celui-ci permettrait de mettre en évidence les différentes caractéristiques de l'équilibre grâce à une visualisation graphique du phénomène.

Le second chapitre traitera de la manière de procéder pour retrouver de manière inductive la relation de GULDBERG-WAAGE. Nous tenterons également de généraliser cette méthode pour une relation $f(v_1, v_2, \ldots v_n) = 0$.

L'analyse fonctionnelle du système fera l'objet du troisième chapitre. Le système y est schématisé dans le but de distinguer les différents composants le constituant.

Le quatrième chapitre traitera de l'analyse organique du système. Celle-ci comprendra notamment la découpe en modules ainsi que la justification de cette découpe.

Dans le cinquième chapitre, nous illustrerons la réutilisabilité des divers modules identifiés au chapitre quatre pour deux nouvelles applications ayant trait à l'équilibre chimique.

Enfin, les spécifications, les détails d'implémentation ainsi que le code de chacun des modules identifiés seront consignés en annexes.

CHAPITRE I

APPROCHE DEDUCTIVE

Comme nous l'avons signalé lors de l'introduction, cette démarche passe par l'élaboration d'un modèle. Nous avons donc développé un modèle à partir duquel il nous sera possible de déduire la loi de GULDBERG-WAAGE.

1.1 Description du modèle

Exemple 1

Considérons que les particules de A et de B peuvent réagir entre elles pour donner les particules de C et de D selon la réaction chimique suivante :

$$A + B \rightleftarrows C + D$$

Supposons que l'on dispose d'un certain volume V contenant un solvant approprié ainsi que :

N_a particules de A;

N_b particules de B;

N_c particules de C;

N_d particules de D.

Ces particules se déplacent sous l'effet de l'agitation thermique.

Faisons l'hypothèse que la réaction $A + B \rightarrow C + D$ est un processus élémentaire (transformation simple).

Pour que cette transformation ait lieu, il faut qu'une particule de A et une particule de B se rencontrent. Cependant, lorsque ces deux particules entrent en collision, elles peuvent :

- soit réagir (on parlera de probabilité P_r d'avoir la réaction);
- soit ne pas réagir.

Le nombre R_r de réactions $A + B \rightarrow C + D$ ayant lieu durant une période d'une seconde est directement proportionnel :

- au nombre N_r de collisions pendant cette période;
- à la probabilité P_r d'avoir la dite réaction.

On peut donc écrire que :

$$R_r = M.N_r.P_r$$

Le nombre de collisions entre particules de A et de B est directement proportionnel :

- au nombre de particules de A;
- au nombre de particules de B.

$$N_r = L.N_a.N_b$$

et donc

$$R_r = k_r.N_a.N_b$$

avec $k_r = f(P_r)$

De même, le nombre R_p de réactions $C + D \rightarrow A + B$ ayant lieu durant une période d'une seconde est égal à

$$R_p = k_p \cdot N_c \cdot N_d$$

avec $k_p = f(P_p)$ où P_p est la probabilité d'avoir la dite réaction.

A l'équilibre, on a que

$$R_r = R_p$$

ou encore que $$k_r \cdot N_a \cdot N_b = k_p \cdot N_c \cdot N_d$$

et donc $$K = \frac{N_c \cdot N_d}{N_a \cdot N_b} = \frac{k_r}{k_p}$$

Il suffit de passer aux concentrations molaires pour retrouver la loi de GULDBERG - WAAGE.

$$K = \frac{[C] \cdot [D]}{[A] \cdot [B]}$$

En effet, sachant qu'il y a N particules dans une mole, il suffit de diviser le nombre de particules d'un composé par le nombre d'Avogadro N pour obtenir la concentration molaire de ce composé.

Exemple 2

Considérons maintenant que la particule de C peut se décomposer pour donner les particules de A et de B selon la réaction chimique suivante :

$$C \rightleftarrows A + B$$

Faisons l'hypothèse que la réaction $C \rightarrow A + B$ est un processus élémentaire.

La particule de C peut :

- soit se décomposer et on partira de la probabilité P_r d'avoir la décomposition;
- soit ne pas se décomposer.

Il semble évident qu'à mesure que le nombre des particules de C augmente , le nombre d'entre elles qui se décomposent dans un intervalle de temps donné augmente également.

Ainsi on aura que :

$$R_r = k_r . N_c \qquad \text{avec } k_r = f(P_r)$$

où R_r = le nombre de réactions de décomposition durant une période d'une seconde.

De même, le nombre R_p de réactions $A + B \rightarrow C$ ayant lieu durant la période d'une seconde est égal à :

$$R_p = k_p . N_a . N_b$$

Comme à l'équilibre $R_r = R_p$

on a que $$k_r . N_c = k_p . N_a . N_b$$

ou encore $$\frac{[A].[B]}{[C]} = K = \frac{k_r}{k_p}$$

Exemple 3

Considérons que les particules de A, de B et de C peuvent réagir entre elles pour donner les particules de D et de E selon la réaction chimique suivante :

$$A + B + C \rightleftarrows D + E$$

En faisant l'hypothèse que la réaction $A + B + C \rightarrow D + E$ est un processus élémentaire, on peut établir selon un raisonnement semblable que :

$$R_r = k_r . N_a . N_b . N_c$$

vu que le nombre de collisions entre les particules de A, de B et de C est directement proportionnel à leur nombre

et que $$R_p = k_p . N_d . N_e$$

A l'équilibre, on a encore $R_r = R_p$

ou $$k_r . N_a . N_b . N_c = k_p . N_d . N_e$$

$$\frac{[D].[E]}{[A].[B].[C]} = K = \frac{k_r}{k_p}$$

Dans les trois exemples mentionnés ci-dessus, nous avons chaque fois fait l'hypothèse que la réaction envisagée était un processus élémentaire (càd se déroulant en une seule étape).

On ne connaît pas de processus élémentaires où plus de trois particules interviennent, car les collisions

simultanées entre plus de trois particules sont très rares.

Mais malheureusement le cours que suivent la plupart des réactions chimiques n'est pas toujours aussi simple.

Exemple 4

Ainsi la réaction $A + 2B + 2C \rightleftharpoons D + 2E$ n'est pas le résultat de la collision simultanée de deux particules de B, de deux particules de C et d'une particule de A.

La probabilité que ces cinq espèces se rencontrent au même moment est très faible, si minime qu'un tel processus ne pourrait jamais s'effectuer à la vitesse observée expérimentalement.

C'est pourquoi de nombreuses réactions peuvent faire intervenir un grand nombre d'étapes. Chaque étape étant un processus élémentaire puisqu'au cours de chacune il se produit une transformation simple.
L'ensemble des processus élémentaires que suit une réaction globale s'appelle mécanisme de la réaction. Le mécanisme des réactions doit être déterminé expérimentalement.

Connaissant le mécanisme de la réaction, il est possible de retrouver la loi de GULDBERG - WAAGE.

Ainsi si

$A + B + C \rightarrow E + F$ ler processus élémentaire

$B + C + F \rightarrow E + D$ 2ème " "

est le mécanisme déterminé expérimentalement de la réaction,

la condition d'équilibre dans un tel système est que

$$R_{r1} = R_{p1}$$

$$R_{r2} = R_{p2}$$

où R_{r1} : le nombre de réactions $A + B + C \rightarrow E + F$

R_{r2} : " $B + C + F \rightarrow E + D$

R_{p1} : " $A + B + C \rightarrow E + F$

R_{p2} : " $B + C + F \rightarrow E + D$

durant la période d'une seconde.

Comme nous l'avons démontré lors de l'exemple trois, on peut donc écrire

$$k_{r1} \cdot [B] \cdot [C] \cdot [A] = k_{p1} \cdot [E] \cdot [F]$$

$$k_{r2} \cdot [B] \cdot [C] \cdot [F] = k_{P2} \cdot [E] \cdot [D]$$

Combinons maintenant ces deux expressions de façon à éliminer $[F]$. En multipliant respectivement les membres de gauche et de droite, nous obtenons :

$$k_{r1} \cdot k_{r2} [B]^2 \cdot [C]^2 \cdot [A] \cdot [F] = k_{p1} \cdot k_{p2} \cdot [E]^2 \cdot [F] \cdot [D]$$

ou encore $$\frac{[E]^2 \cdot [D]}{[B]^2 \cdot [C]^2 \cdot [A]} = K = \frac{k_{r1} \cdot k_{r2}}{k_{p1} \cdot k_{p2}}$$

1.2 Simulation d'une réaction chimique

Afin de mieux se familiariser avec le concept d'équilibre chimique, il serait intéressant de mettre au point un programme de simulation.

Celui-ci permettrait, par exemple, de mettre en évidence :

- l'influence du nombre de particules sur le nombre de collisions entre ces particules;
- l'influence du nombre de particules sur la fréquence de réaction;
- l'influence de la probabilité d'avoir la réaction sur la fréquence de cette réaction;
- l'aspect dynamique de l'équilibre et de la réversibilité des réactions chimiques;
- la possibilité d'avoir des réactions en une ou plusieurs étapes.

L'ordinateur, grâce à ses grandes capacités graphiques, reste un outil privilégié. Une façon de procéder, par exemple, serait de faire apparaître à l'écran des particules représentées sous la forme de figures géométriques. Ces particules se déplaceraient aléatoirement dans un volume délimité par les parois d'un récipient. Ces particules pouvant rentrer en collision, réagir, se dissocier,... selon le cas envisagé.

On disposerait également de plusieurs compteurs permettant à tout instant de comptabiliser :

- le nombre de particules d'une espèce donnée.
- le nombre de collisions entre les particules de réactifs ou de produits.
- le nombre de réactions effectives ayant eu lieu entre les divers produits en présence.
- ...

CHAPITRE II

ANALYSE INDUCTIVE

Les valeurs de concentration à l'équilibre ne peuvent pas être quelconques car alors il n'y aurait pas de réaction possible entre les constituants en présence. Il existe donc une relation entre les concentrations à l'équilibre de ces différents constituants. L'objectif de ce chapitre est de développer une façon de procéder pour retrouver de manière inductive cette relation.

Comment retrouver une relation existant entre plusieurs variables? Nous allons tout d'abord illustrer la marche à suivre sur un exemple et ensuite nous la généraliserons.

2.1 Exemple

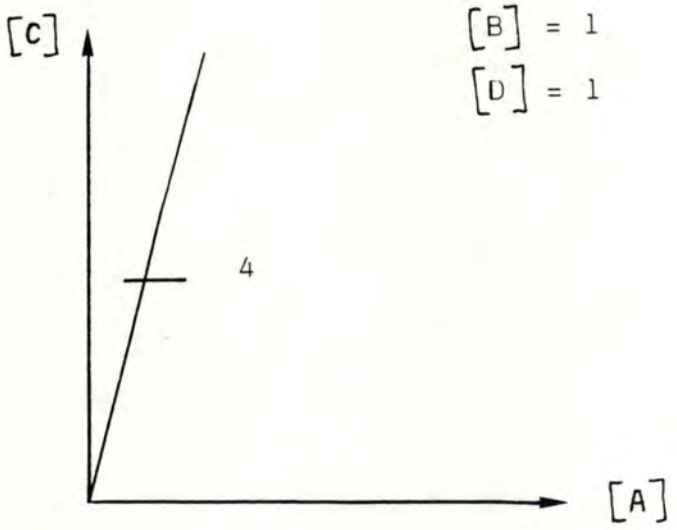
Considérons que les quatres composés A,B,C et D peuvent réagir selon la réaction chimique suivante

$$A + 2B \rightleftharpoons C + 2D$$

Nous allons essayer de trouver la relation qui existe entre les concentrations à l'équilibre de ces 4 composés. Afin de simplifier l'écriture, on représentera la concentration à l'équilibre d'un composé X par $[X]$.

Une façon de procéder, par exemple, consiste à étudier la variation de $[C]$ lorsqu'on fait varier $[A]$ en laissant $[B]$ et $[D]$ constantes. (Graphique 1)

Pour cet exemple particulier, nous conviendrons que 4 points suffisent pour pouvoir tracer une courbe représentative de la variation relative de 2 variables quelconques.



Graphique 1.

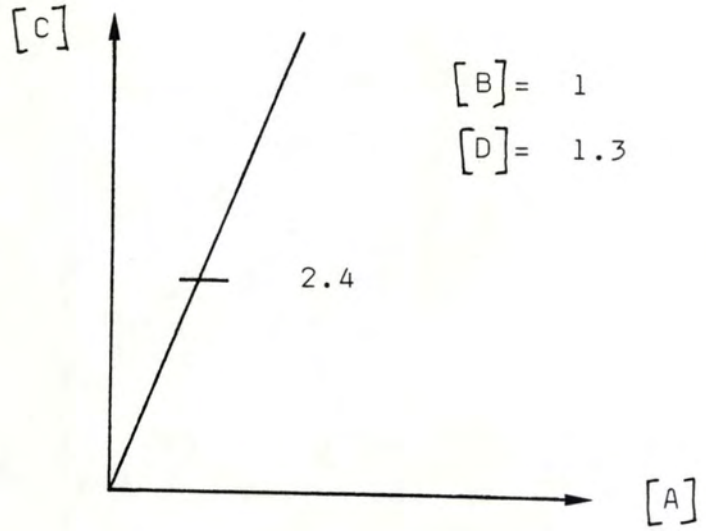
On s'aperçoit très vite qu'il existe une relation linéaire entre [C] et [A] . Toutefois, nous avons considéré que [B] et [D] étaient des constantes alors qu'elles sont en fait des variables.

Que se passe-t-il si l'on modifie la [B] et/ou [D] maintenues constantes? Nous allons donc procéder à d'autres expériences dans le but de pouvoir par la suite étudier l'influence de [B] et [D] .

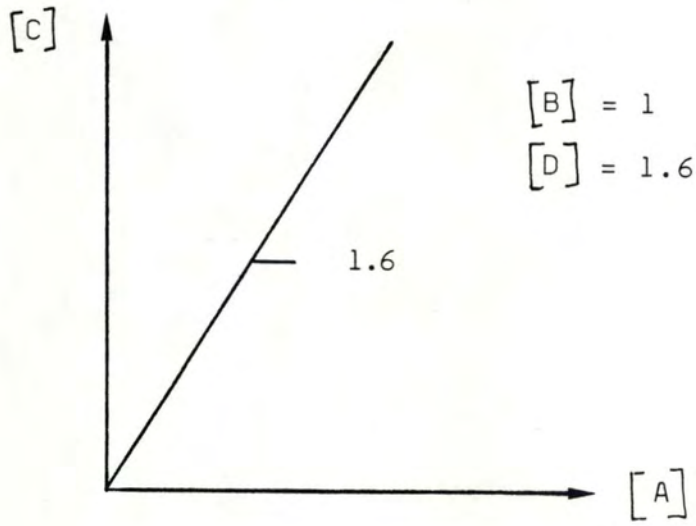
On remarquera que les valeurs de [B] et [D] ne sont pas choisies au hasard. Il s'agit en fait d'une série de couples ([B],[D]) où l'on fait varier [D] de façon identique pour différentes valeurs de [B] .

(Graphiques 2 à 16)

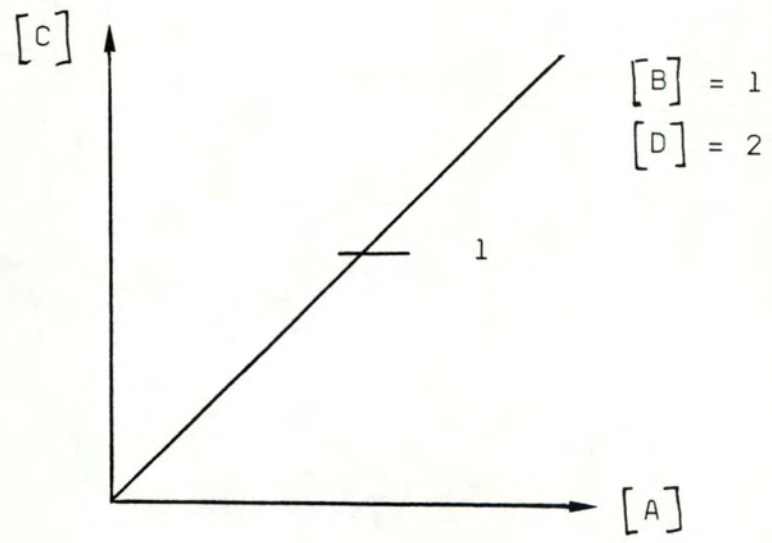
Graphique 2



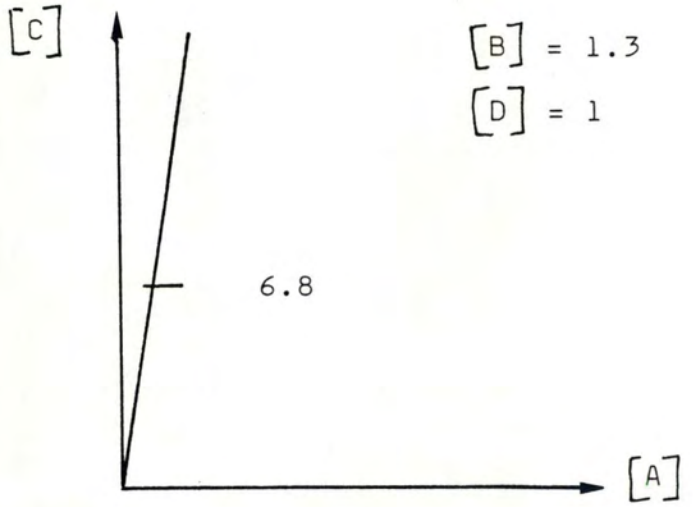
Graphique 3



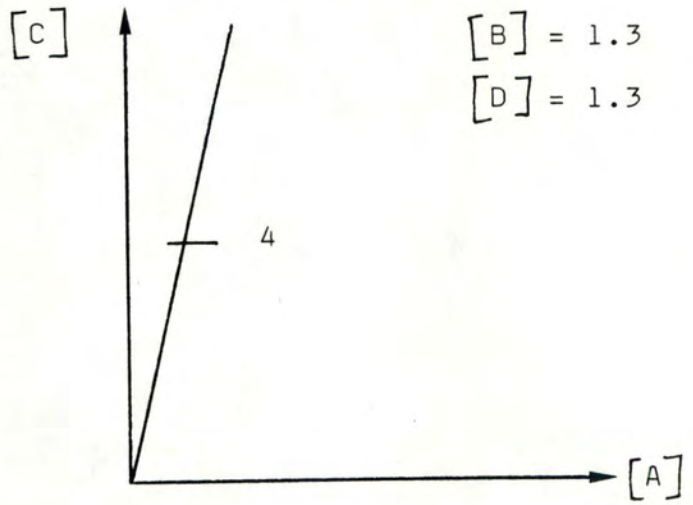
Graphique 4



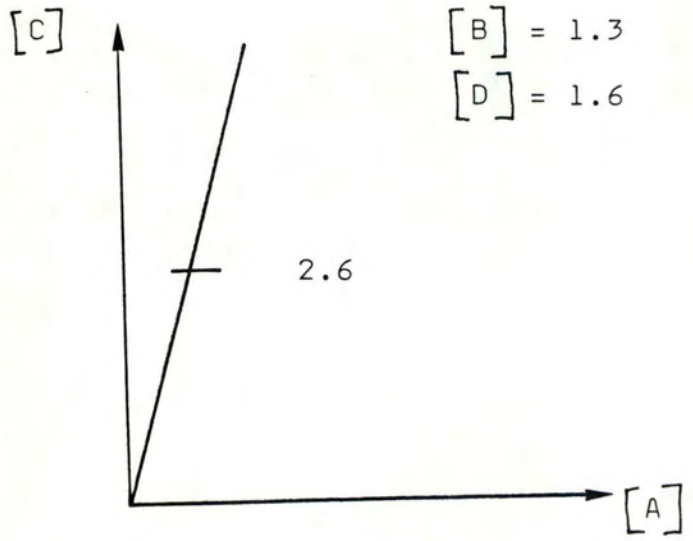
Graphique 5



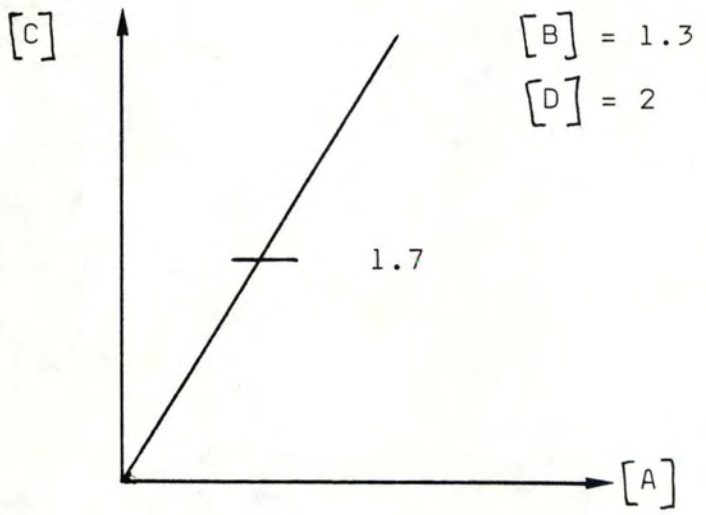
Graphique 6



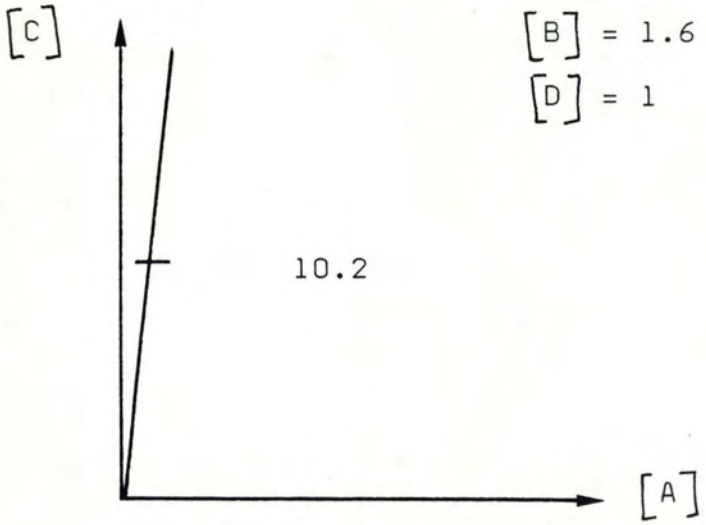
Graphique 7



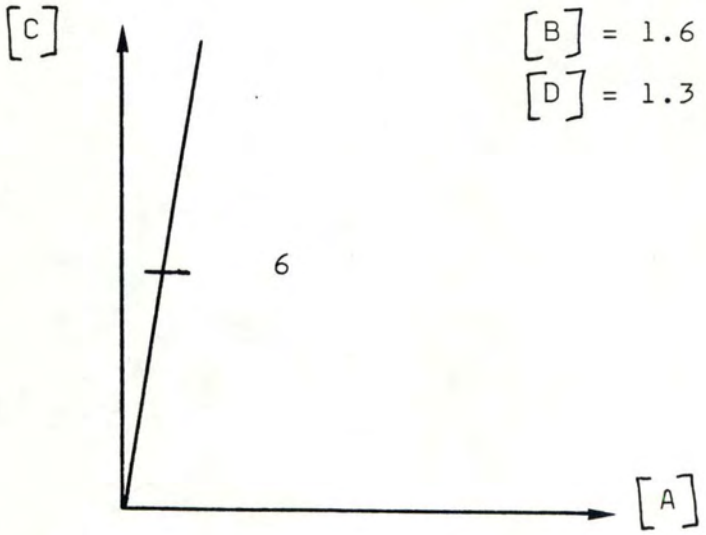
Graphique 8



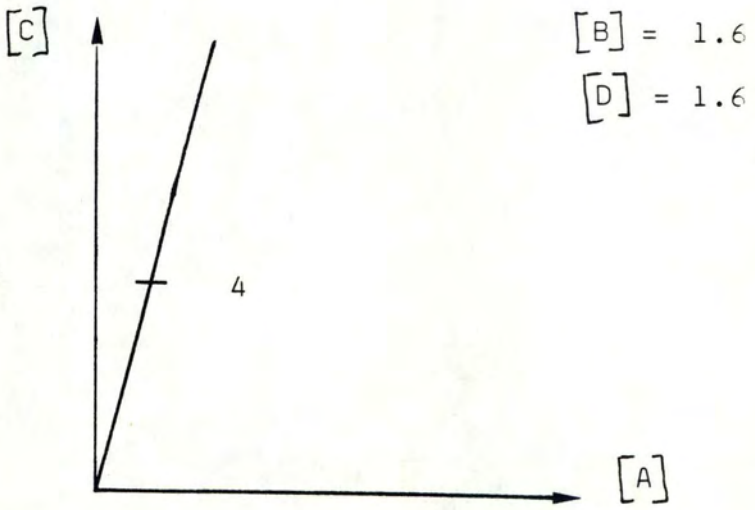
Graphique 9



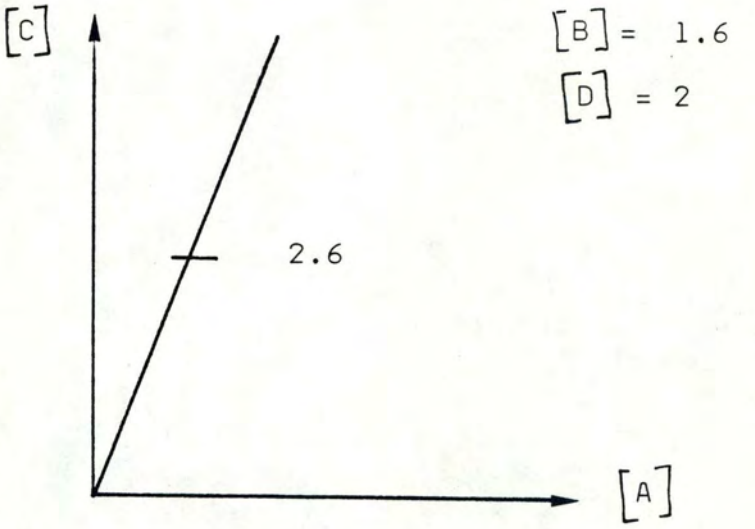
Graphique 10



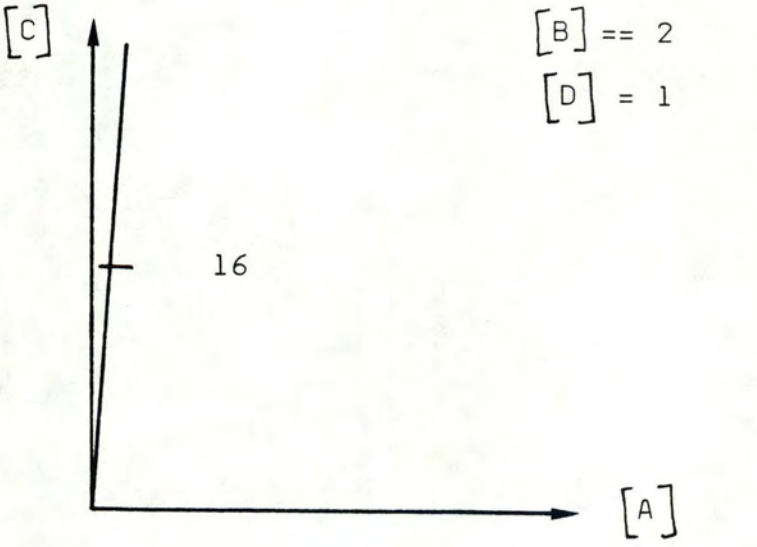
Graphique 11



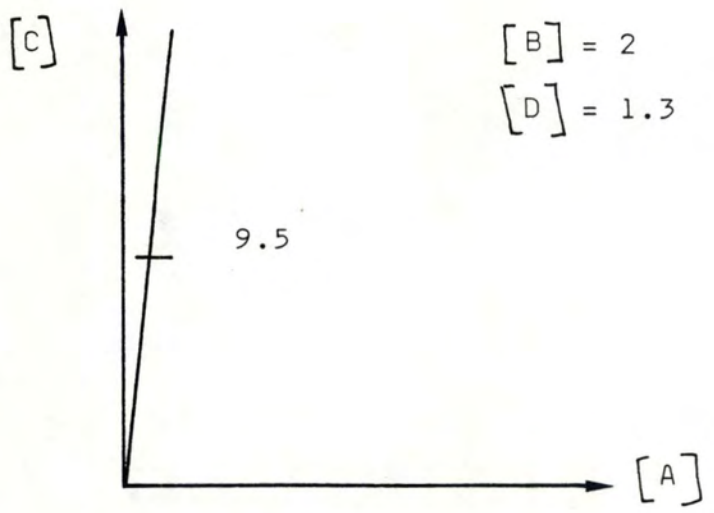
Graphique 12



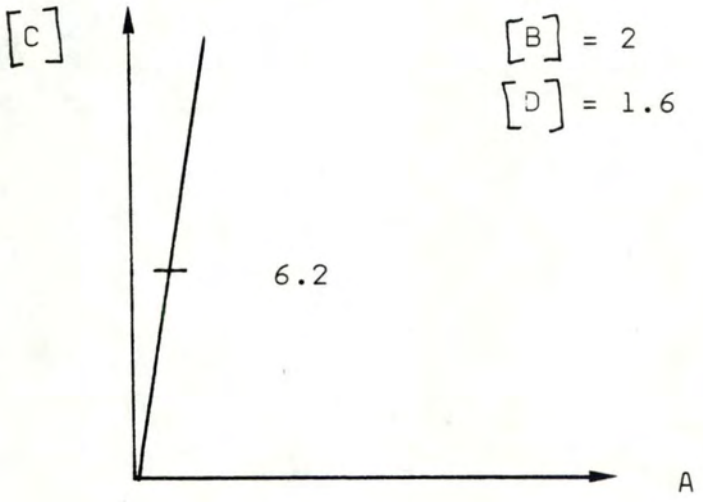
Graphique 13



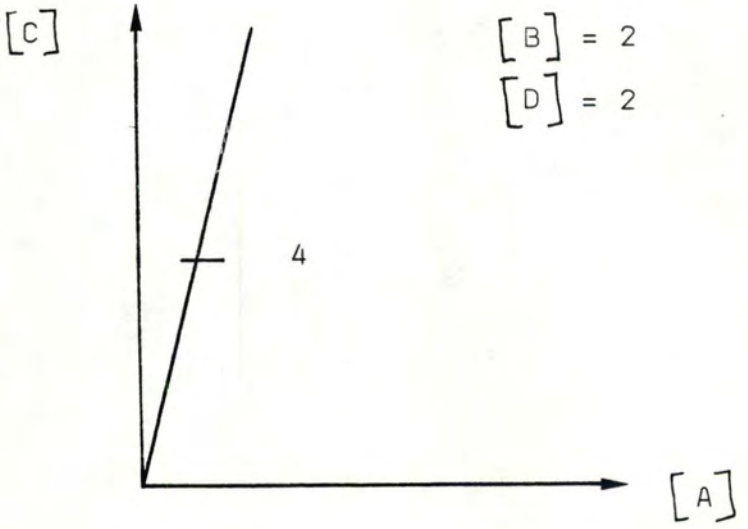
Graphique 14



Graphique 15



Graphique 16



On s'aperçoit, à posteriori, qu'en ayant choisi un système d'axes orthonormés, on obtient, dans tous les cas, une droite passant par les quatre points.

L'équation générale de cette droite peut s'écrire :

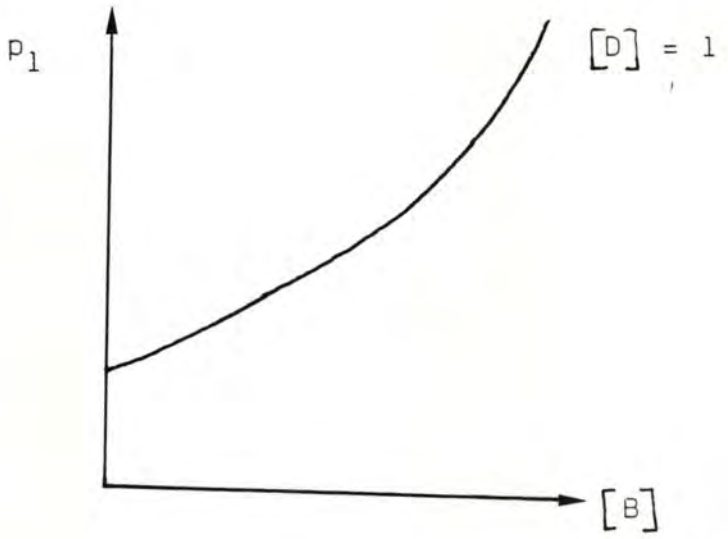
$$C = p_1 \cdot [A] + o_1$$

avec $p_1, o_1 = f([B], [D])$.

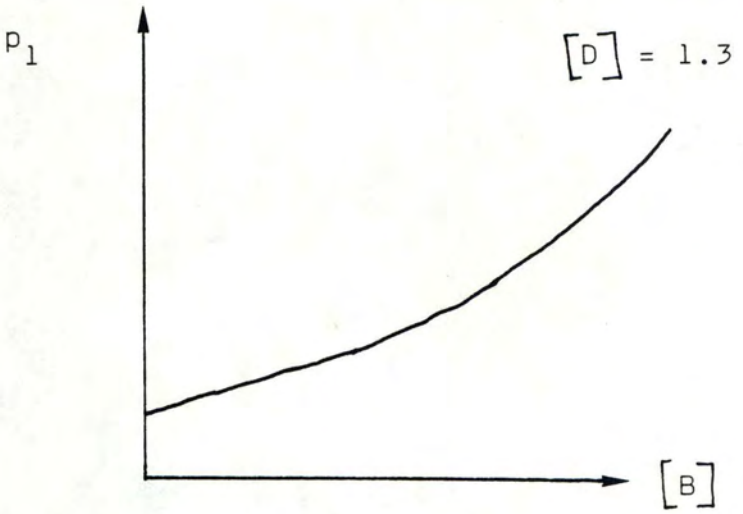
On notera que le nombre d'expériences que l'on a dû réaliser est égal à $4^{(4-1)} = 64$. En effet, il faut à chaque fois quatre points pour pouvoir tracer la courbe représentative de la variation relative de deux variables et il faut déterminer la variation de $[C]$ en fonction de la variation des trois autres concentrations à l'équilibre.

Nous allons maintenant déterminer comment la pente "p_1" et l'ordonnée à l'origine "o_1" varient en fonction de $[B]$ et $[D]$. Pour ce faire, nous allons, par exemple, étudier séparément la variation de p_1 (graphiques 17 à 20), et la variation de o_1 (graphiques 25 à 28) en fonction de $[B]$ en laissant $[D]$ constante.

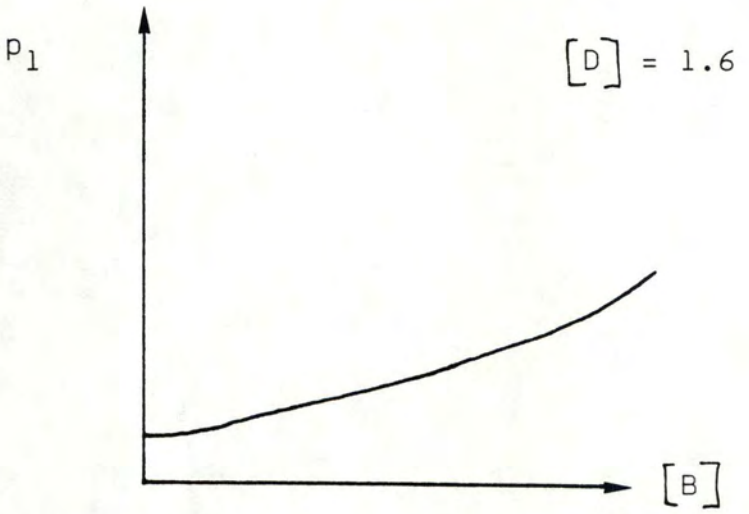
Graphique 17



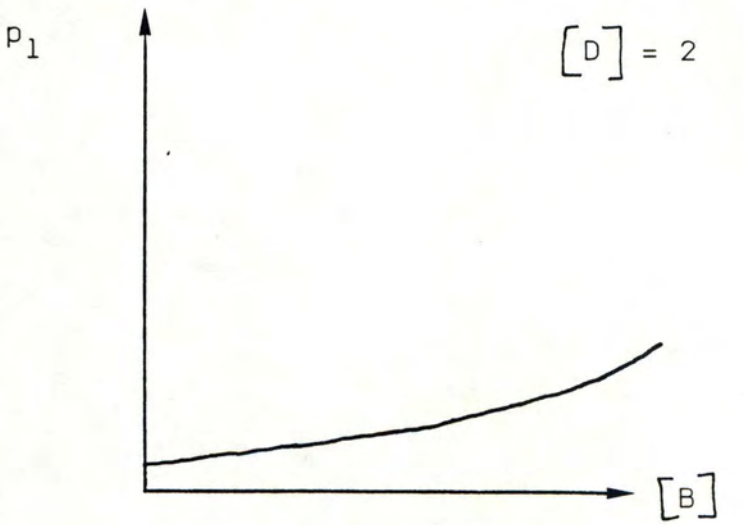
Graphique 18



Graphique 19



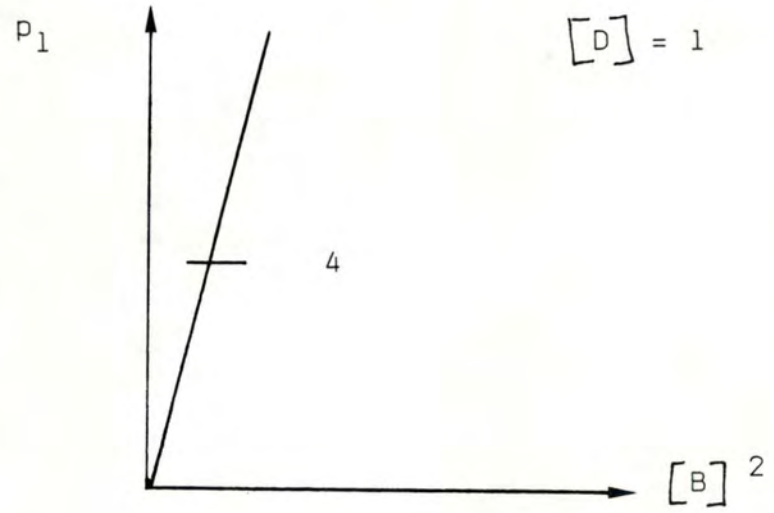
Graphique 20



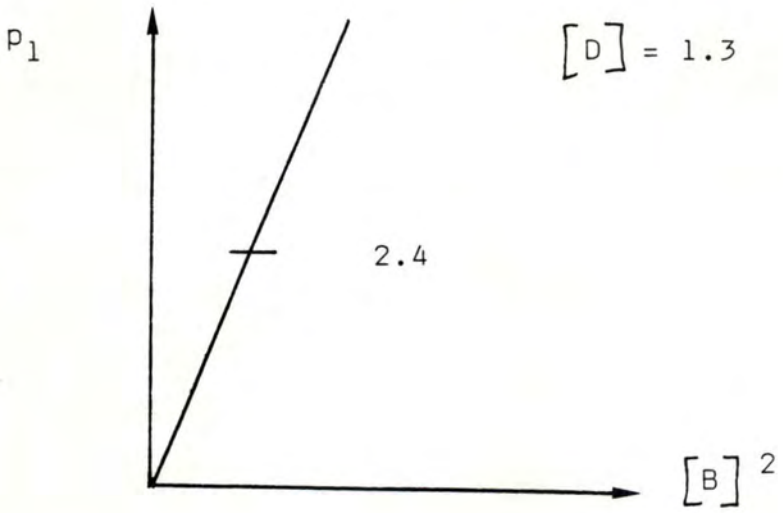
On remarque que la courbe représentative de la variation de p_1 en fonction de la variation de $[B]$ est dans les quatre cas une parabole.

Afin d'obtenir une relation linéaire entre ces deux variables, nous allons plutôt étudier la variation de p_1 en fonction de la variation de $[B]^2$ (graphiques 21 à 24).

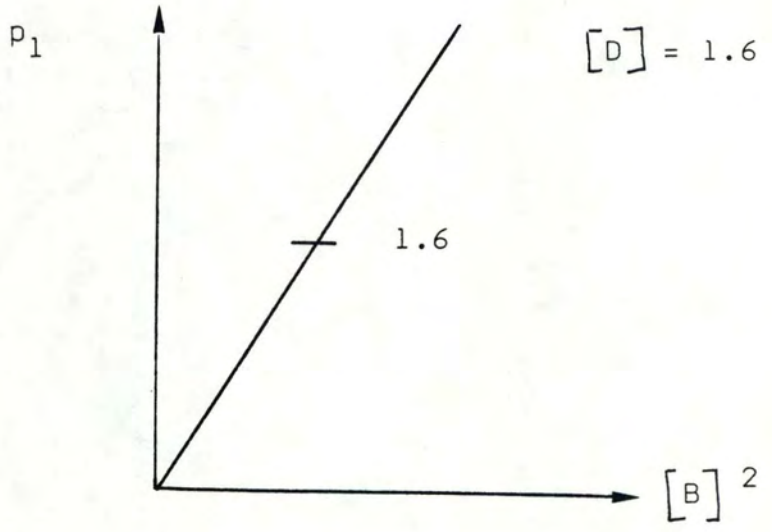
Graphique 21



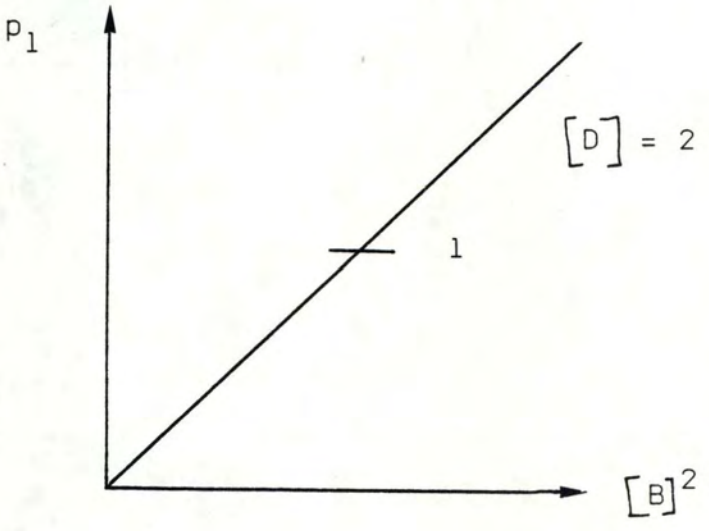
Graphique 22



Graphique 23



Graphique 24

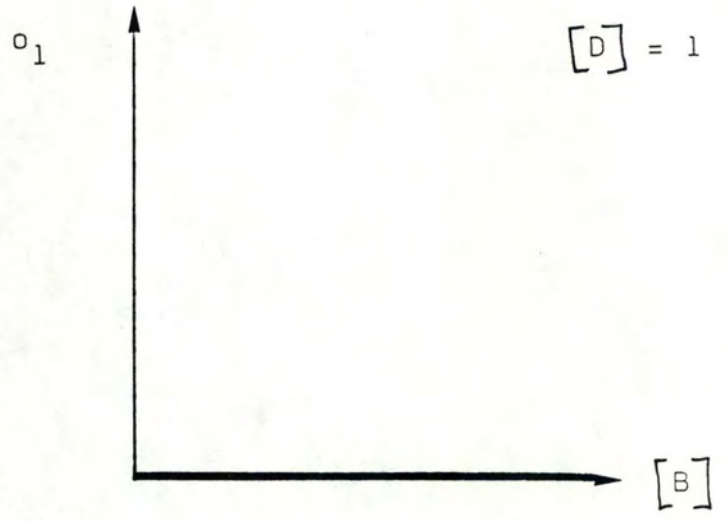


Dans les quatre cas, on obtient une droite dont l'équation générale peut s'écrire :

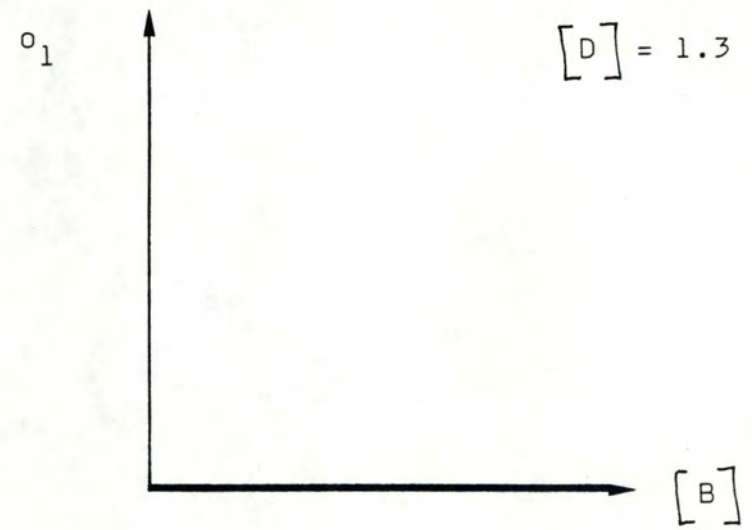
$$p_1 = p_2 \cdot [B]^2 + o_2$$

avec p_2, $o_2 = f([D])$.

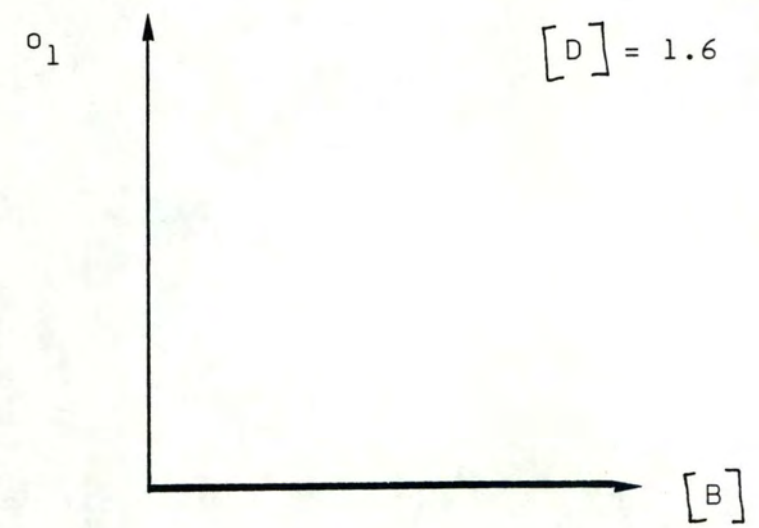
Graphique 25



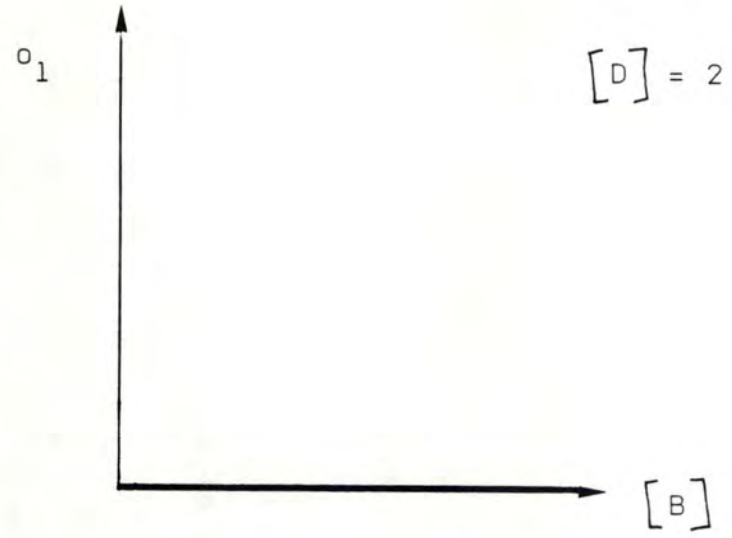
Graphique 26



Graphique 27



Graphique 28



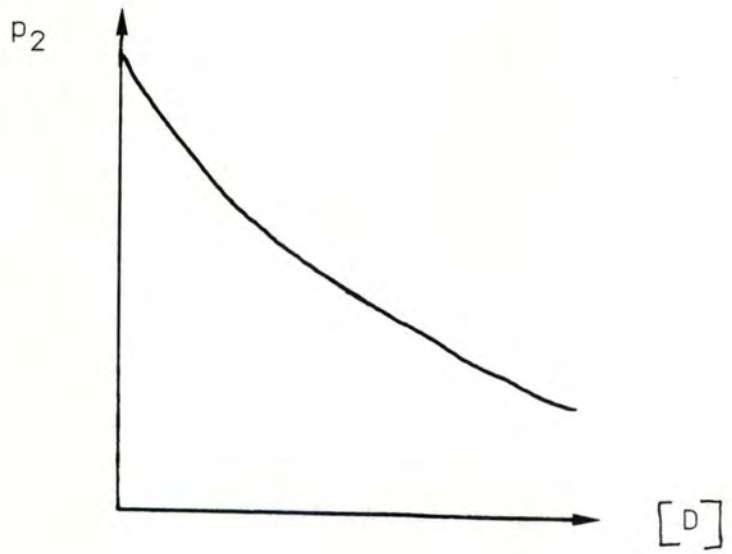
On déduit de l'étude de la variation de o_1 en fonction de la variation de $[B]$ l'équation générale :

$$o_1 = p_3 \cdot [B] + o_3$$

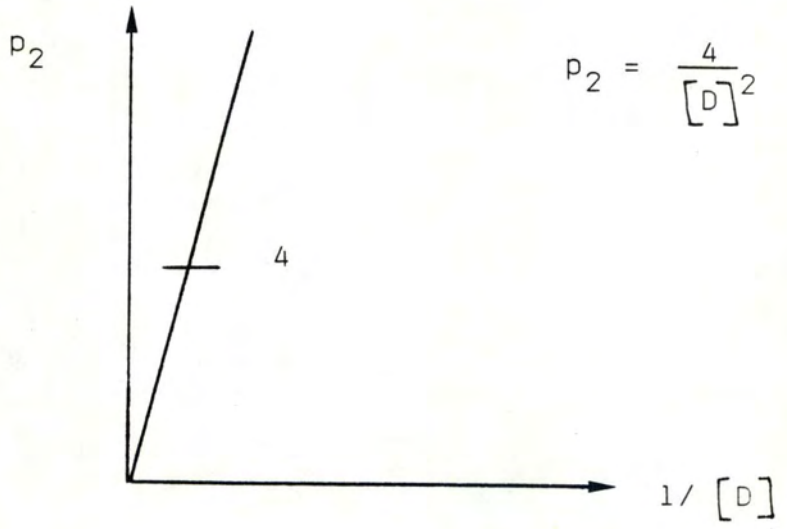
avec p_3, $o_3 = f([D])$.

Il faut enfin déterminer si p_2, o_2, p_3 et o_3 sont fonction de $[D]$. Pour ce faire, on étudiera séparément leur variation en fonction de la variation de $[D]$ (graphiques 29 à 33).

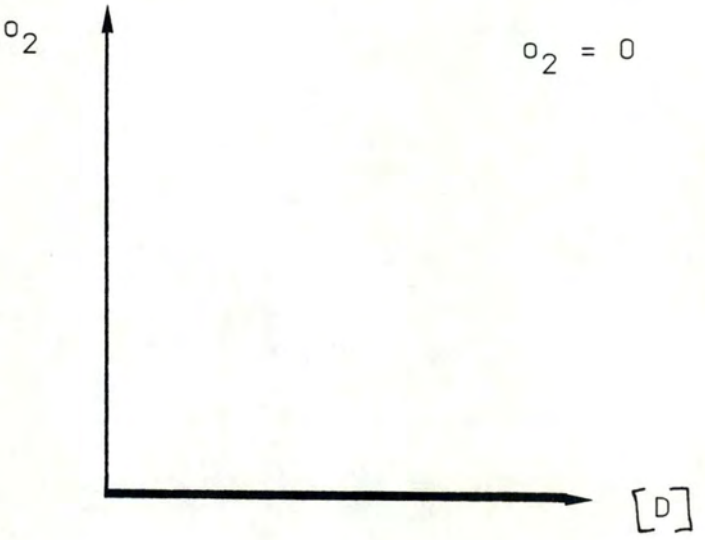
Graphique 29



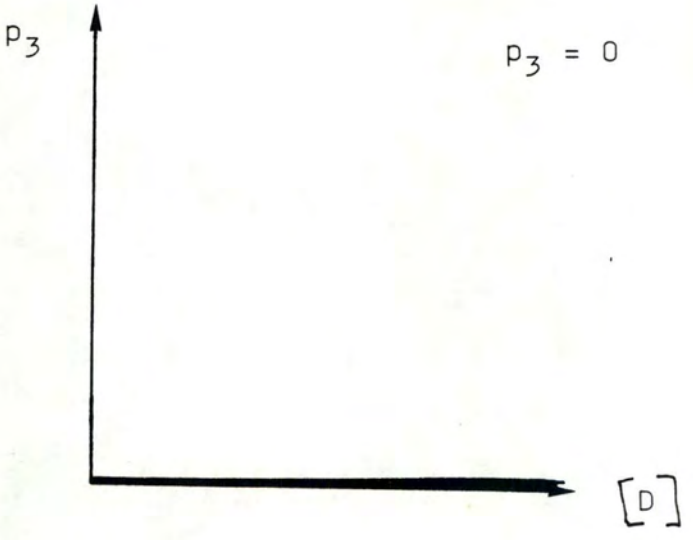
Graphique 30



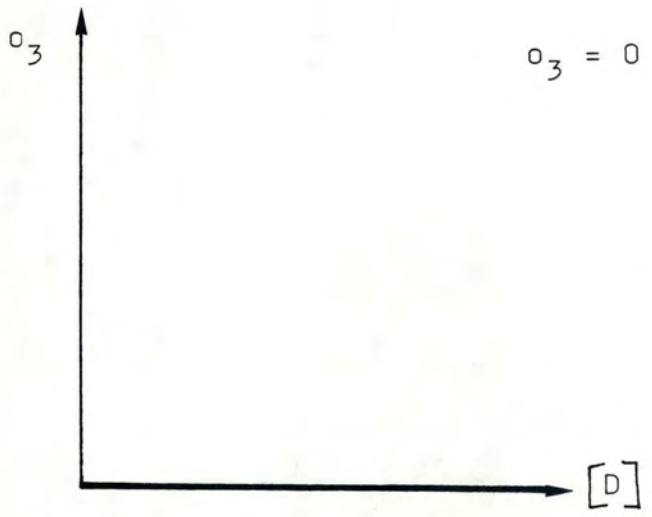
Graphique 31



Graphique 32



Graphique 33



La pente et l'ordonnée à l'origine des différentes droites obtenues ne sont maintenant plus fonction d'aucune variable .

Nous savons donc que :

$$[C] = p_1 \cdot [A] + o_1$$

$$p_1 = p_2 \cdot [B]^2 + o_2 \qquad o_1 = p_3 \cdot [B] + o_3$$

$$p_2 = 4/[D]^2 \qquad o_2 = \emptyset \qquad p_3 = \emptyset \qquad o_3 = \emptyset$$

Grâce à ces $2^{4-1}-1=7$ équations, il nous est possible de formuler la relation existant entre les quatre variables pour le domaine de variation des concentrations à l'équilibre choisi ci-dessus.

$$C = p_2 \cdot [B]^2 + \cdot A + o_2 + p_3 \cdot B + o_3$$

$$C = \frac{4\,[B]^2.[A]}{[D]^2} + \emptyset + \emptyset.[B] + \emptyset$$

$$\frac{[C].[D]^2}{[B]^2.[A]} = 4$$

On s'aperçoit donc que le produit des concentrations du membre de droite, chacune élevée à une puissance égale à son coefficient stoechiométrique, sur le produit des concentrations du membre de gauche, chacune élevée à une puissance égale à son coefficient stoechiométrique, est égale à une constante qui en l'occurence vaut 4.

Remarquons que la relation trouvée peut encore s'écrire :

$$[C].[D]^2 - 4\,[B]^2.[A] = \emptyset$$

Nous avons donc retrouvé la relation $f([A], [B], [C], [D]) = \emptyset$ entre les 4 concentrations à l'équilibre.

Nous allons maintenant tenter de généraliser la marche à suivre pour n variables.

2.2 Généralisation

Supposons que l'on désire trouver une relation $f(v_1, v_2, \ldots v_n) = \emptyset$ existant entre n variables. Dans ce cas, on étudiera la variation d'une première variable "v_1" lorsqu'on fait varier une seconde variable "v_2" en laissant les n-2 dernières variables constantes.

Afin de pouvoir étudier par la suite l'influence de ces n-2 variables, ce type d'expérience sera répété en faisant varier séparément chacune de ces n-2 variables de façon analogue à celle décrite dans l'exemple mentionné ci-dessus.

Si l'on considère que n_p est le nombre de points nécessaires pour pouvoir représenter graphiquement la variation relative de 2 variables, le nombre de points nécessaires pour trouver la relation est égal à $n_p^{(n-1)}$.

Ce nombre représente donc le nombre d'expériences qu'il y aura lieu de réaliser.

Notons que le nombre de graphiques v_1/v_2 est pour sa part égal à $n_p^{(n-2)}$.

Il s'agit alors de choisir les échelles du système d'axes de façon à ce qu'une droite passe par les n_p points dans les $n_p^{(n-2)}$ cas. On aura ainsi trouvé une relation linéaire entre les 2 variables v_1 et v_2.

L'équation générale de la droite représentative de cette relation linéaire peut s'écrire :

$$v_1' = p_1 . v_2' + o_1$$

avec $v_2' = f(v_2)$

$v_1' = f(v_1)$

et $p_1, o_1 = f$ (n-2 variables).

Il faut ensuite déterminer comment la pente p_1 et l'ordonnée à l'origine o_1 sont fonction des n-2 variables maintenues constantes. Pour ce faire, on étudiera séparément la variation de o_1 et p_1 en fonction d'une

de ces n-2 variables, soit "v_3", en laissant les n-3 autres constantes.

Afin de pouvoir étudier l'influence de ces n-3 variables, ce type d'expérience sera répété en faisant varier séparément chacune de ces n-3 variables de façon analogue à celle décrite dans l'exemple.

On essayera encore de choisir les échelles du système d'axes de façon à ce qu'une droite passe par les np points dans les $(np)^{n-3}$ cas.

On pourra alors écrire :

$$p_1' = p_2 . v_3' + o_2$$

avec $p_1' = f(p_1)$

$v_3' = f(v_3)$

$p_2, o_2 = f$ (n-3 variables)

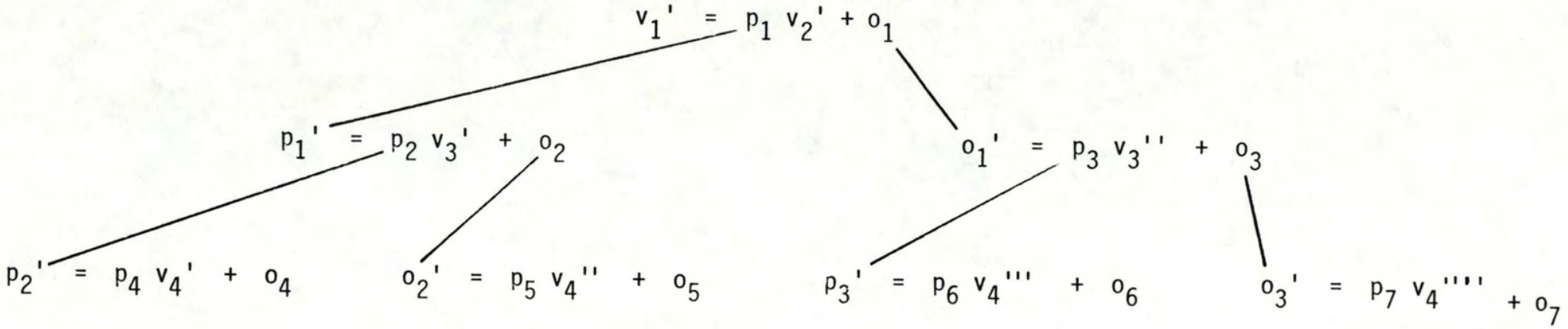
et $$o_1' = p_3 . v_3'' + o_3$$

avec $o_1' = f(o_1)$

$v_3'' = f(v_3)$

$p_3, o_3 = f$ (n-3 variables)

Ce processus sera répété ainsi de suite jusqu'à ce que la pente et l'ordonnée à l'origine ne soient plus fonction d'aucune variable. On disposera alors de $2^{n-1} - 1$ équations. En les combinant, on retrouvera l'expression de la relation existant entre les n variables :


$v_1' = p_1 v_2' + o_1$
$p_1' = p_2 v_3' + o_2$
$o_1' = p_3 v_3'' + o_3$
$p_2' = p_4 v_4' + o_4$
$o_2' = p_5 v_4'' + o_5$
$p_3' = p_6 v_4''' + o_6$
$o_3' = p_7 v_4'''' + o_7$

....

Remarques :

1. Le cas n = 1 n'a aucune signification dans cette généralisation et le cas n = 2 est trivial.

2. Nous avons déterminé que le nombre d'expériences que l'on doit réaliser pour trouver la relation existant entre les n variables est égal à n_p^{n-1}.
   On s'aperçoit très vite que cette façon de procéder n'est pratiquement réalisable que si le nombre de variables que comporte la relation n'est pas trop élevé.

CHAPITRE III

ANALYSE FONCTIONNELLE

Disposant maintenant d'une méthode pour retrouver de façon inductive la loi de GULDBERG-WAAGE, nous devons encore être capable :

- de pouvoir introduire l'équation chimique pour laquelle on désire étudier la relation existant entre les concentrations à l'équilibre;
- de pouvoir déterminer des valeurs de concentration à l'équilibre, point de départ de notre démarche inductive.

3.1 Présentation du langage

L'objectif de ce paragraphe est de définir le formalisme nécessaire pour pouvoir introduire interactivement une équation chimique.

Il faut donc convenir du vocabulaire et de la syntaxe càd du langage qui sera utilisé par l'utilisateur pour décrire l'équation à l'ordinateur. Le langage dictera quelles expressions seront "comprises" par l'ordinateur. Ce langage se voudra aussi proche que possible de celui communément employé par les chimistes.

Cependant, les caractères disponibles sur le terminal influenceront quelque peu la syntaxe.

Cette solution permet à l'élève de pouvoir utiliser le langage auquel il est habitué.

En effet, à ce stade de leurs études, les élèves ont déjà étudié cette notion d'équation chimique. Cette façon d'introduire l'énoncé peut donc servir de test pour voir s'ils ont bien assimilé cette notion.

De plus, le concept de réactivité chimique intervient non seulement dans l'enseignement de l'équilibre chimique mais dans beaucoup d'autres domaines encore. Le même langage pourra donc être réutilisé dans de nombreux contextes différents. Ce qui ne fait qu'augmenter l'intérêt qu'on peut lui porter.

Nous donnons ci-après une description d'une équation chimique calquant au mieux celle qu'on utilise dans l'enseignement secondaire.

Une équation chimique est constituée de deux membres que nous appellerons "membre de gauche" et "membre de droite". Ces deux membres sont séparés soit par une flèche orientée vers la droite, si la réaction envisagée est irréversible, soit par deux flèches opposées, si la réaction envisagée est réversible (ou équilibrée).
Le membre de gauche reprend la liste des réactifs et le membre de droite la liste des produits de la réaction. Chacun de ces deux membres est en fait une liste de composés. Ces derniers sont éventuellement précédés d'un coéfficient stoechiométrique.

Un composé peut être soit un ion, soit un atome ou encore une molécule. Lorsque le coéfficient stoechiométrique d'un composé vaut 1, il peut être éventuellement omis mais en aucun cas il ne pourra valoir zéro. Une molécule est un assemblage d'atomes dans des proportions bien définies en accord avec les règles de valence . Un ion est soit un atome chargé, soit une molécule chargée. Dans ce cas, la charge est indiquée au-dessus de l'ion en question.

Exemple :

$$NaOH + HCl \rightarrow NaCl + H_2O$$

$$Ba(OH)_2 + H_2SO_4 \rightarrow BaSO_4 + 2\ H_2O$$

$$N_2 + 3\ H_2 \rightleftharpoons 2\ NH_3$$

$$Ag^+ + Cl^- \rightleftharpoons AgCl$$

Il conviendra également d'effectuer une analyse sémantique de l'équation introduite. Celle-ci consiste premièrement à vérifier que l'équation chimique est stoechiométriquement équilibrée càd qu'il y a le même nombre d'atomes de réactifs que d'atomes de produits.

Deuxièmement, il faut vérifier que l'équilibre électrique est lui aussi respecté càd que la charge totale des réactifs est la même que celle des produits.

Enfin, les composés introduits par l'élève doivent correspondre à des composés chimiquement stables, en accord avec les règles de valence et d'étage d'oxydation.

Il est cependant très difficile de déterminer, grâce à ces règles, si un composé chimique est stable ou non.
Cette difficulté réside principalement dans le fait qu'un atome peut posséder plusieurs étages d'oxydation. Ceux-ci étant fonction de son entourage. Vu l'impossibilité de développer une méthode simple fournissant un résultat exact dans tous les cas, nous avons intentionnellement omis ce type de vérification.

Remarques :

- Notons toutefois que le langage développé ne permet pas d'appréhender toutes les équations chimiques envisageables. En effet, en utilisant les formules brutes pour identifier un composé, il nous est impossible de distinguer deux composés différents mais ayant une même formule moléculaire càd des isomères.

- Nous avons intentionellement oublié de traiter les cas exceptionnels des radicaux ($CH_3^{\bullet}$) ou encore des molécules hydratées ($CaCl_2.6H_2O$). Ces composés étant rarement rencontrés dans les exemples prisés par les enseignants du secondaire.

3.2 Obtention des concentrations à l'équilibre

Soit la réaction suivante :

$$cs_1 c_1 + \ldots + cs_{nr} c_{nr} \rightleftarrows cs_{nr+1} c_{nr+1} + \ldots + cs_{nc} c_{nc}$$

où :

- nc : nombre de composés;
- nr : " de réactifs;
- c_i : le composé i;
- cs_i : coéfficient stoechiométrique du composé i.

La constante d'équilibre de la réaction peut s'écrire comme suit :

$$K = \frac{\text{produit } [i : nr + 1 .. nc] \ ([c_i])^{cs_i}}{\text{produit } [j : 1 .. nr] \ ([c_j])^{cs_j}}$$

où $[c_i]$ est la concentration à l'équilibre du composé i.

Il suffit donc d'attribuer une valeur de concentration à l'équilibre pour nc-1 composés, la valeur de la concentration à l'équilibre du dernier composé étant automatiquement fixée par la loi de GULDBERG -WAAGE.

Ainsi, si la dernière concentration est celle d'un produit c_i ($nr+1 <= i <= nc$) alors :

$$([c_i])^{cs_i} = \frac{K * \text{produit } [j:1 .. nr] \ ([c_j])^{cs_j}}{\text{produit } [l:nr+1.. nc; l \neq i] \ ([c_l])^{cs_l}}$$

Si la dernière concentration est celle d'un réactif c_i ($1 <= i <= nr$) alors :

$$([c_i])^{cs_i} = \frac{\text{produit } [j:nr+1 .. nc] \ ([c_j])^{cs_j}}{K * \text{produit } [l:1 .. nr; l \neq i] \ ([c_l])^{cs_l}}$$

Pour que ce calcul soit envisageable, il faut disposer de la valeur de la constante d'équilibre.

L'obtention de cette valeur fera l'objet du paragraphe suivant.

Afin de mieux rendre compte de la réalité expérimentale dans la détermination des concentrations à l'équilibre, il serait bon de confronter l'élève avec le fait qu'une expérience est toujours entachée d'une erreur expérimentale. En effet, il est essentiel que ce dernier ne prenne pas l'habitude de travailler avec des données de précision illimitée.

3.3 Calcul de la valeur de la constante d'équilibre

Comme nous venons de le voir dans le paragraphe précédent, l'obtention des concentrations à l'équilibre nécessite la connaissance de la valeur de la constante d'équilibre.

Une première solution serait de demander à l'utilisateur, qu'après avoir introduit l'équation, il donne la valeur de la constante d'équilibre. Cette solution simple possède cependant l'inconvénient majeur d'obliger à connaître la valeur de cette constante . C'est pourquoi nous avons préféré calculer de façon tout à fait interne la valeur de celle-ci.

Nous allons maintenant donner une description de la méthode préconisée.

La thermodynamique a montré que $K = \exp(-\Delta G^0 / RT)$ où :

- K:constante d'équilibre;
- ΔG^0 : variation d'énergie libre standard de la réaction;

- R : constante de Boltzmann;
- T : température.

On sait également que $\Delta G^0 = \Delta H^0 - T.\Delta S^0$

où:

- ΔH^0 : variation d'enthalpie de la réaction;
- ΔS^0 : " d'entropie " " " .

Donc, $K = \exp(-\Delta H^0 / RT) * \exp(\Delta S^0 / R)$.

Il ne reste donc qu'à savoir calculer le ΔH^0 et le ΔS^0 pour la réaction envisagée. Pour ce faire, on utilisera les valeurs tabulées des enthalpies de formation (H_f^0) et des entropies absolues (S_f^0).

En effet, on a que :

- ΔH^0 = somme H_f^0 (produits) - somme H_f^0 (réactifs)
- ΔS^0 = " S_f^0 (") - somme S_f^0 (")

Cette méthode ne permet pas toujours d'obtenir une valeur de la constante d'équilibre pour plusieurs raisons :

1. il n'est pas toujours possible d'obtenir une valeur de l'enthalpie de formation ou de l'entropie absolue d'un composé car ces valeurs sont impossibles à obtenir expérimentalement.
2. cette méthode exige que l'on mémorise la valeur de l'enthalpie de formation et de l'entropie absolue des composés. Trois cas de figure sont envisageables :

a. le composé dont on recherche le H^0_f et le S^0_f est un composé chimiquement stable et est repris dans la liste des composés mémorisés;

b. le composé dont on recherche le H^0_f et le S^0_f est un composé chimiquement stable mais n'est pas repris dans la liste des composés mémorisés;

c. le composé dont on recherche le H^0_f et le S^0_f est un composé non stable chimiquement (donc inexistant).

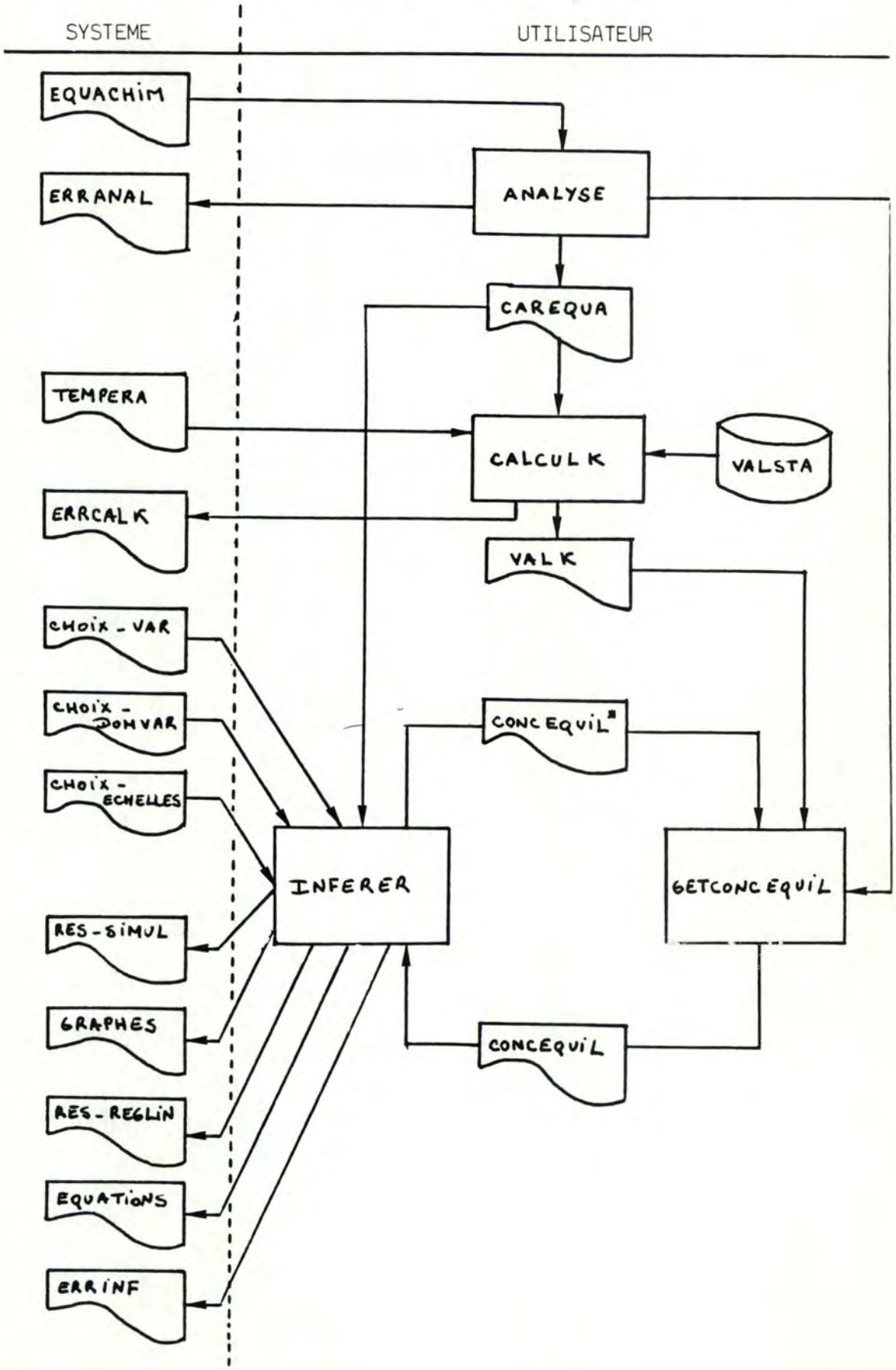
Dans les deux derniers cas, il nous est impossible de calculer la valeur de la constante d'équilibre. Aussi prendra-t-on soin de le signaler à l'utilisateur par l'envoi d'un message approprié. Celui-ci spécifiera que l'on est dans l'impossibilité de continuer car on est en présence d'un composé "inconnu" du système. Ce composé "inconnu" pouvant correspondre à un composé stable mais non mémorisé ou à un composé inexistant (ou instable).

Remarque :

- Si la liste des composés mémorisés ne contient que des composés chimiquement stables, on a réussi à faire une vérification partielle de l'existence des composés chimiques intervenant dans l'équation introduite par l'utilisateur. En effet, si le calcul de la valeur de la constante d'équilibre aboutit, on est donc en présence d'une équation dont tous les composés sont chimiquement stables.

3.4 Schéma global du système

Nous allons maintenant distinguer les différents composants du système (schéma 1).



L'utilisateur introduit interactivement l'équation chimique selon le langage défini ci-dessus (EQUACHIM). Le système procède alors à l'analyse syntaxique et sémantique de cette équation. Au terme de celle-ci, le système a identifié les différentes caractéristiques de l'équation (CAREQUA).

En cas d'erreurs, la nature et l'emplacement de la première d'entre elles sont présentés à l'utilisateur (ERRANAL).

L'utilisateur introduit ensuite la température à laquelle la réaction a lieu (TEMPERA). Le système calcule alors la valeur de la constante d'équilibre (VALK). Pour ce faire, il utilise les valeurs standards qu'il a mémorisées (VALSTA). Si toutefois ce calcul n'aboutit pas, un message d'erreur est présenté à l'utilisateur (ERRCALK).

Enfin, le système guide l'utilisateur dans la façon de procéder pour retrouver de manière inductive la loi de GULDBERG-WAAGE selon la méthode explicitée au chapitre II.

Tout au long de cette méthode, l'utilisateur aura plusieurs choix à effectuer. Ainsi il devra :

- choisir les variables en ordonnée et en abscisse (CHOIX - VAR);
- fixer le domaine de variation de chacune des variables (CHOIX - DOMVAR);
- sélectionner les échelles du système d'axes de façon à obtenir une relation linéaire lors de l'étude de la variation relative de 2 variables (CHOIX - ECHELS).

Le système, pour sa part, facilitera la tâche de l'utilisateur en :

- simulant les résultats des diverses expériences (RES - SIMUL);
- effectuant les transformations adéquates des données selon le choix des échelles effectué par l'utilisateur et réalisant les différents graphiques (GRAPHES);
- réalisant les régressions linéaires (RES - REGLIN);
- formulant les équations des droites représentatives des relations linéaires obtenues (EQUATIONS).

Enfin, au cas où l'utilisateur aurait effectué un mauvais choix d'échelles du système d'axes, un message d'erreur serait présenté à l'utilisateur (ERRINF).

Remarque :

- Comme nous l'avons signalé précédemment, la méthode préconisée pour le calcul de la valeur de la constante d'équilibre nécessite la connaissance de différentes valeurs standards. Celles-ci devront donc être mémorisées par le système. On devra donc développer un programme permettant l'enregistrement de ces valeurs standards ainsi que leur consultation, modification ou encore suppression.

3.4.1 Définition des traitements

Nous allons maintenant décrire brièvement les 3 composants suivant le modèle ci-après :

- nom du composant;
- messages en entrée;
- messages en sortie;
- informations mémorisées et/ou consultées;
- sa fonction.

1. Analyse

Reçoit : message externe : equachim.
message interne : /.
Produit : message interne : carequa.
message externe : erranal.

Fonction : si equachim est correcte alors carequa.
sinon erranal.

2. Calcul de K

Reçoit : message externe : tempera.
message interne : carequa.
Produit : message interne : valk.
message externe : errcalk.
Informations consultées : valsta.
Fonction : si le calcul de K est possible alors : valk.
sinon : errcalk.

3. Inférer

Reçoit : message interne : concequil, carequa.

message externe : choix-varao, choix-echels, choix-domvar.

Fournit : message externe : res-simul, res-reglin, graphes, équations, errinf.

message interne : concequil.

Fonction : guider l'utilisateur dans la façon de procéder pour retrouver de façon inductive la loi de GULDBERG-WAAGE;

4. Getconcequil

Reçoit : message externe : /.

message interne : valk, carequa, concequil.

Fournit : message externe : /.

message interne : concequil.

Fonction : calcul de concentrations à l'équilibre.

3.4.2 Définition des données

equachim : équation chimique formulée dans le langage défini au chapitre II.

erranal : nature et emplacement de la première erreur rencontrée lors de l'analyse;

carequa : caractéristiques de l'équation correcte càd :

- la formule brute, la charge et le coefficient stoechiométrique des différents composés intervenant dans l'équation;
- le nombre de réactifs (nr) et de composés (nc) que comprend l'équation;
- le type de réaction envisagée (directe ou équilibrée).

tempera : température à laquelle a lieu la réaction;

valk : valeur de la constante d'équilibre;

errcalk : message signalant la non connaissance des valeurs standards d'un composé;

valsta : valeurs standards nécessaires au calcul de la valeur de la constante d'équilibre :

- l'enthalpie de formation (H_f^0);
- l'entropie absolue (S_f^0);

concequil* : valeur des concentrations à l'équilibre pour nc-1 composés (nc est le nombre de composés que comporte l'équation);

concequil : valeur des concentrations à l'équilibre pour les nc composés.

choix-domvar : le domaine de variation des différentes variables càd :

- valeur minimale;
- valeur maximale;

que peut prendre chacune des variables;

choix-echels : fonction de transformation à appliquer aux échelles du système d'axes;

choix-var : identificateur d'une variable pouvant figurer en abscisse ou ordonnée dans l'étude de la variation relative de 2 variables;

res-simul : résultats de l'étude de la variation relative de 2 variables;

res-reglin : résultats de la régression linéaire :

- les paramètres a,b et c de la droite de régression $a.x+b.y+c = 0$

- le coefficient de corrélation.

graphes : graphiques représentatifs de la variation relative de 2 variables;

équations : expression de l'équation de la droite représentative de la relation linéaire existant entre 2 variables;

errinf : message d'erreur signalant que la relation envisagée n'est pas linéaire dans tous les cas.

CHAPITRE IV

ANALYSE ORGANIQUE

Nous allons maintenant procéder à la découpe en modules de notre système. D'une part, cette découpe a pour objectif de faciliter les différentes étapes du développement de notre application (conception, spécification, codage, validation et maintenance). D'autre part, cette découpe doit également essayer de satisfaire les différentes exigences d'un bon didactitiel (modifiabilité, réutilisabilité, extensibilité, portabilité et documentabilité).

4.1 Découpe en modules

Nous avons identifié 3 modules. Nous allons dans un premier temps donner une brève description de chacun des modules. Les spécifications, algorithmes ainsi que le code sont consignés en annexes.

A. Module chimique

A.1 Module "analyseur-chimique"

Celui-ci est composé de 2 fonctions qui ont pour but d'analyser une chaîne de caractères introduite interactivement par l'utilisateur pour :

- déterminer si celle-ci représente bien une équation chimique syntaxiquement et sémantiquement correcte;
- déterminer si celle-ci représente un composé chimique syntaxiquement correct .

A.2 Module "fonctions - chimiques"

Celui-ci reprend toute une série de fonctions chimiques telles que :

- calcul de la constante d'équilibre;
- calcul de l'énergie libre de réaction;
- calcul de l'enthalpie de réaction;
- calcul de l'entropie de réaction;
- calcul de l'énergie libre des produits et des réactifs;
- calcul de l'enthalpie des produits et des réactifs;
- calcul de l'entropie des produits et des réactifs;
- ...

A.3 Module "valeurs - standards"

Ce module de données permet la mémorisation des valeurs standards relatives à un composé chimique.

La manipulation de ces valeurs standards se fera par l'intermédiaire de primitives telles que :

- enregistrement de nouvelles valeurs standards;
- consultation séquentielle ou spécifique des valeurs standards;
- modification de valeurs standards mémorisées;
- suppression de valeurs standards mémorisées;
- détermination de la mémorisation des valeurs standards d'un composé chimique.

B Module démarche inductive

B.1 Module "inferer"

Ce module permet de trouver de manière inductive une relation $f(v_1, v_2, \ldots, v_n) = 0$ existant entre n variables selon la méthode développée au chapitre III.

B.2 Module "fonction - inferer"

Ce module reprend, pour chaque relation $f(v_1, v_2, \ldots, v_n) = 0$ susceptible d'être retrouvée inductivement, une fonction qui a pour but de simuler les résultats d'expériences nécessaires pour retrouver la relation en question ainsi qu'une procédure qui a pour but de fixer le domaine de variation des différentes variables.

B.3 Module utilitaire

B.3.1 Module "utilitaire - graphes"

Ce module permet de dessiner le graphique représentatif de la variation relative de 2 variables ainsi que la droite de régression sur le graphique en question.

B.3.2 Module "utilitaire - reglin"

Ce module permet d'effectuer une régression linéaire :

- de y en x;
- de x en y;
- orthogonale selon le choix.

B.3.3 Module"utilitaire - transfo - echels"

Ce module permet la transformation des échelles du système d'axes. Les diverses transformations possibles sont consignées dans un menu à choix multiples.

B.3.4 Module"utilitaire - E/S"

Ce module reprend toute une série de fonctions dont l'objectif est de faciliter les entrées-sorties. On retiendra principalement la saisie d'un entier et d'un réel.

C Module coordinateur

Ce module est responsable de la séquence d'appel des différents modules suivant le scénario établi lors de l'analyse fonctionnelle.

4.2 Justification de la découpe en modules

Nous avons choisi de localiser dans 3 modules distincts :

- les aspects chimiques de l'application;
- la démarche inductive proprement dite;
- le scénario d'exécution.

Les raisons principales sont que :

- la démarche inductive pourrait très bien être appliquée à une autre relation du type $f(v_1, v_2, \ldots, v_n) = 0$ que celle de GULDBERG-WAAGE. Le module "DEMARCHE_INDUCTIVE" pourrait donc être réutilisé dans un contexte tout à fait différent du contexte chimique.
- le scénario étant localisé dans un seul module pourra facilement être modifié ou adapté aux désirs et besoins de l'utilisateur.

Le module "CHIMIQUE" a été subdivisé en 3 sous - modules :

- un module "analyseur - chimique" :

Comme nous l'avons déjà signalé, les chimistes utilisent un langage pour décrire les phénomènes chimiques qu'ils étudient. Il était donc opportun de mettre au point un module permettant à un utilisateur d'utiliser le même langage pour décrire ces problèmes chimiques .

- un module "fonctions-chimiques":

L'idée est qu'on pourrait développer progressivement une librairie de fonctions chimiques. L'avantage d'une telle librairie est que le programmeur pourra utiliser les diverses fonctions disponibles sans avoir à les programmer.

- un module "valeurs_standards":

L'idée est qu'on pourrait constituer, moyennant quelques aménagements, une base de données chimiques où, pour un composé chimique particulier, on disposerait de ses caractéristiques physiques, chimiques, thermodynamiques, ...

Le module " démarche inductive " a lui aussi fait l'objet d'une découpe plus fine:

- nous avons localisé dans le module "fonctions à inférer" les différentes relations qu'on pourra retrouver grâce à l'algorithme d'induction afin de rendre ce dernier le plus indépendant possible de ces relations.

- L'utilisation des fonctions regroupées dans les modules "utilitaires" permet de rendre l'algorithme d'induction plus clair et plus lisible. De plus, ces différentes fonctions pourront être réutilisées dans un grand nombre d'applications nouvelles.

CHAPITRE V

ILLUSTRATION DE LA REUTILISABILITE DES MODULES

Afin de montrer la réutilisablité des différents modules établis précédemment nous avons décidé de développer deux applications supplémentaires.

La première consistera à découvrir la loi de Van't Hoff de façon inductive. (Etude de la variation de la constante d'équilibre en fonction de la température).
La seconde consistera à étudier l'influence de la température sur la réversibilité des réactions (réactions exothermiques et endothermiques).

5.1 Variation de la constante d'équilibre en fonction de la température

Nous avons déjà vu que :

$$K = \exp(-\Delta G^0/RT)$$

On peut donc écrire que :

$$\ln K = -\frac{\Delta G^0}{RT}$$

En dérivant par rapport à T, on a que :

$$\frac{d(\ln K)}{dT} = \frac{d(-\Delta G^0/RT)}{dT}$$

$$\text{ou} \quad \frac{d(\ln K)}{d\,T} = \left(-\frac{DGO}{R}\right) \cdot \frac{d(1/T)}{d\,T} + \left(\frac{-1}{RT}\right) \cdot \frac{d(DGO)}{d\,T}$$

$$\text{or} \quad DGO = DHO - T \cdot DSO$$

$$\text{donc} \quad \frac{d\,(\ln K)}{d\,T} = \frac{DGO}{RT^2} + \frac{T}{T} \cdot \frac{(DSO)}{RT} = \frac{DHO}{RT^2}$$

à condition que DHO ne soit pas fonction de la température T.

En intégrant, on a que :

$$\ln K = \text{cste} - \frac{DHO}{RT}$$

Donc si on étudie la variation de lnK en fonction de 1/T on obtient une droite dont la pente est égale à $-\frac{DHO}{R}$.

Voyons maintenant comment les différents modules identifiés lors de l'analyse organique pourront être réutilisés dans cette nouvelle application.

On utilisera exactement les mêmes modules. Cependant, il faudra prendre soin :

- d'ajouter dans le module "fonction à inférer" la fonction qui permettra de calculer la valeur de la constante d'équilibre pour une valeur de température donnée;
- de permettre l'appel à cette fonction dans le module "inférer";
- de modifier le scénario au niveau du coordinateur.

5.2 Influence de la température sur la réversibilité des réactions

1. Si la pente de la droite obtenue en étudiant la variation de ln K en fonction de la variation de 1/T est négative, cela signifie que DHO est positif et que la réaction est endothermique.

Donc, si on augmente la température, la constante d'équilibre augmente càd que la concentration de chacun des produits augmente et celle de chacun des réactifs diminue. On en tire une première conclusion : si la réaction est endothermique alors une augmentation de température favorise les produits.

2. Si la pente de la droite obtenue en étudiant la variation de ln K en fonction de la variation de 1/T est positive, cela signifie que DHO est négatif et que la réaction est exothermique.

Donc, si on augmente la température, la constante d'équilibre diminue càd que la concentration de chacun des produits diminue et celle de chacun des réactifs augmente. On en tire une seconde conclusion : si la réaction est exothermique alors une augmentation de la température favorise les réactifs.

Comment illustrer ces 2 conclusions ?

Soit une réaction :

$$CS_1\ C_1 + \ldots\ CS_{nr}C_{nr} \rightleftarrows CS_{nr+1}C_{nr+1} + \ldots + CS_{nc}C_{nc}$$

où nr : nombre de réactifs

nc : nombre de composés

C_j : composé j

CS_j : coefficient stoechiométrique du composé j

Supposons qu'initialement :

pour tout $1 \leqslant j \leqslant nr$: $[C_j]\, i > 0$

pour tout $nr + 1 \leqslant j \leqslant nc$: $[C_j]\, i = 0$

où $[C_j]\, i$ est la valeur de la concentration initiale pour le composé j.

A l'équilibre on a que :

pour tout $1 \leqslant j \leqslant nr$: $[C_j]\, e = [C_j]\, i - CS_j * x$

pour tout $nr + 1 \leqslant j \leqslant nc$: $[C_j]\, e = CS_j * x$

où $[C_j]\, e$ est la valeur de la concentration à l'équilibre pour le composé j.

En effet, une fraction des réactifs a réagi stoechiométriquement pour donner les produits de la réaction selon l'équation mentionnée ci-dessus.

Ecrivons la constante d'équilibre :

$$K = \frac{\text{produit } [\, j : nr + 1 \ldots nc\,] \; (CS_j * x)^{CS_j}}{\text{produit } [\, k : 1 \ldots nr\,] \; (C_k\, i - CS_k * x)^{CS_k}}$$

Pour obtenir les valeurs des concentrations à l'équilibre, il suffit donc de rechercher la valeur de "x" càd une des racines du polynôme suivant :

$$F(x) = K * \text{produit}\left[k:1 \,..\, nr\right] \left(\left[C_k\right] i - CS_k * x\right)^{CS_k} - \text{produit}\left[j:nr+1 \,..\, nc\right] \left(CS_j * x\right)^{CS_j} = 0$$

en sachant que $0 < x < \min\left(\left[C_j\right] i / CS_j\right)$ pour tout $1 \leqslant j \leqslant nr$ vu qu'on ne peut pas consommer plus que ce qui est disponible.

Il reste donc à visualiser sur un graphique la variation de la concentration à l'équilibre d'un réactif ou d'un produit lorsqu'on fait varier la température, ceci en partant chaque fois de concentrations initiales en réactif identiques.

Voyons quels modules pourront encore être réutilisés dans cette seconde application.

- l'utilisateur introduit l'équation chimique grâce au langage développé au chapitre I.(Module "ANALYSEUR-CHIMIQUE");

- l'évaluation de la valeur de la constante d'équilibre à différentes températures est directement réalisable grâce aux fonctions du module "FONCTIONS-CHIMIQUES". Toutefois, il faudra développer une nouvelle fonction. Celle-ci aura pour but de déterminer, à une température donnée, les concentrations à l'équilibre des différents composés à partir des concentrations initiales en réactifs. Cette fonction sera bien entendu ajoutée à celles du module "FONCTIONS-CHIMIQUES";

- la visualisation graphique [réactif ou produit]e/T sera réalisée grâce à la primitive du module "GRAPHES";

- le module "COORDINATEUR" devra pour sa part être complètement révisé.

Conclusions:

Nous concluons ce travail en formulant deux souhaits :

1. Nous espérons que ce programme sera effectivement utilisé comme support à l'enseignement de l'équilibre chimique.
   D'une part, cela permettrait de voir à quel point les objectifs ont été atteints. En particulier, il serait intéressant de savoir si la compréhension de l'élève en a été modifiée.
   D'autre part, une utilisation du didactitiel par les élèves eux-mêmes ferait sûrement apparaître certaines lacunes ou certaines modifications à apporter.

2. Nous espérons également que ce travail connaîtra des prolongements. Il y a certainement des améliorations et des extensions qu'une utilisation du didactitiel sur le terrain devrait faire apparaître.

Nous allons maintenant énumérer les améliorations et les extensions auxquelles nous avons d'ores et déjà pensé.

Améliorations

1. Dans le système actuel, chacune des équations de droite représentative d'une relation linéaire existant entre 2 variables est directement affichée à l'écran par le système.
   On pourrait envisager une version où l'utilisateur

introduirait interactivement chacune de ces équations. Chacune de celles-ci serait alors comparée à celle déterminée par le système.
En cas d'erreur, l'équation correcte serait alors proposée à l'utilisateur.

2. Le module "analyseur-chimique" ne nous permet pas actuellement de garantir l'existence d'un composé chimique. Il serait donc commode de développer une méthode rapide et efficace permettant d'affirmer à coup sûr si un composé chimique est stable ou non.

3. Pour obtenir une relation linéaire entre 2 variables, nous avons choisi de modifier les échelles du système d'axes. Pour ce faire, l'utilisateur sélectionne, dans un menu à choix multiples, la fonction de transformation adéquate. Le menu regroupe les fonctions "classiques" généralement disponibles sur une calculette de poche. Cette solution, simple à réaliser possède cependant plusieurs inconvénients :

   - elle limite le nombre de fonctions de transformation envisageables;
   - elle limite l'esprit de création de l'élève. En effet, celui-ci ne peut choisir que les fonctions proposées dans le menu;
   - elle ne permet pas toujours d'obtenir une relation linéaire.

C'est pourquoi, on pourrait envisager que ce soit l'utilisateur lui-même qui introduise interactivement

la fonction de transformation. Cette solution, plus complexe dans sa réalisation, permettrait à l'utilisateur de combiner les fonctions "classiques" selon son propre gré.

4. Enfin, on pourrait envisager de combiner les différentes équations établies lors de la démarche inductive de façon à obtenir une expression simplifiée au maximum de la relation recherchée.

Extensions

1. D'une part, on pourrait développer d'autres types d'exercices relatifs au concept d'équilibre chimique. Ainsi, on pourrait :

   - pour les différents composés d'une équation donnée :
     * déterminer les concentrations à l'équilibre à partir des concentrations initiales en réactif.
     * retrouver les concentrations initiales en réactif à partir des concentrations à l'équilibre.
   - envisager la détermination de valeurs de pH, produit de solubilité, degré d'ionisation,...

2. D'autre part, il pourrait être intéressant de savoir générer des équations chimiques. Pour ce faire, il faudrait :

- un moyen d'obtenir des composés chimiques stables;
- un moyen efficace d'obtenir des réactions chimiques "possibles". En effet, n'importe quel composé chimique ne réagit pas avec n'importe quel autre;
- un moyen d'équilibrer électriquement et stoechiométriquement l'équation engendrée;

3. Enfin et surtout, il faudrait réaliser le programme de simulation de l'équilibre chimique. Rappelons que quelques idées quant à la manière de procéder ont déjà été émises lors du premier chapitre.

BIBLIOGRAPHIE

1. DE MONTMOLLIN, M. : "L'enseignement programmé", Presses universitaires de France (Que sais-je ?), Paris, 1975.

2. HEBENSTREIT, J. : L'enseignement assisté par ordinateur : où en est-on ?", 01 Informatique, janvier-février 1979, n° 127, pp32-38.

3. HOUZIAUX, M. : "Vers l'enseignement assisté par ordinateur", Presses universitaires de France, Paris, 1972.

4. P.W. ATKINS : "Physical Chemistry", Oxford University Press, Oxford, 1978.

5. B.M. MAHAN : "Chimie", Interéditions, Paris, 1977.

6. P. DAGNELIE : "Théorie et Méthodes statistiques" Presses agronomiques de Gembloux, Vol 1.

7. R. GUIMUR : "Procédures de tri", MASSON, 1983.

8. K. JENSEN et N. WIRTH : "Pascal Manuel de l'utilisateur" Eyrolles, Paris, 1982.

9. Apple PASCAL : Operating System Reference Manual.
Apple PASCAL : Apple PASCAL LANGUAGE Reference Manual.

ANNEXES

1. Spécifications et Algorithmes 77

I Module Chimique 77

A Module "Analyseur-Chimique"..... 77

B Module "Valeurs-Standards"..... 108

C Module "Fonctions Chimiques"..... 122

II Module Démarche Inductive 136

A Module "Inférer"..... 136

B Module "Fonctions-à-Inférer"..... 169

C Module "Utilitaire-Graphes"..... 174

D Module "Utilitaire -E/S" 187

E Module "Utilitaire-Reglin"..... 194

F Module "Utilitaire-Echelles"..... 198

III Module "coordinateur" 203

2. Architecture physique 208

1 Découpe en modules physiques 208

2 Architecture physique proprement dite 208

3 Codage et Intégration 210

1. SPECIFICATIONS ET ALGORITHMES.

I. MODULE CHIMIQUE

A Module "ANALYSEUR-CHIMIQUE"

1. Fonctions de l'interface

a. Fonction EQUACHIM

Résultat : - vrai si la suite de caractères introduite interactivement par l'utilisateur correspond à une équation chimique syntaxiquement et sémantiquement correcte alors :

- NC = le nombre de composés que comporte l'équation;
- NR = le nombre de réactifs que comporte l'équation;
- TYRE = le type de réaction identifié.

pour tout $1 \leqslant i \leqslant NC$:

- TCO [i] = l'identificateur du ième composé identifié dans l'équation;
- TCS [i] = le coefficient stoechiométrique du composé TCO [i] ;
- TCH [i] = la charge électrique du composé TCO [i] .

Postcondition : - pour tout $1 \leqslant i \leqslant NR$:

TCO [i] = un réactif.

- pour tout $NR+1 \leqslant i \leqslant NC$:

TCO [i] = un produit.

- $NC > NR$.

- faux sinon alors :

- message d'erreur.

b. Fonction COMPCHIM

Résultat : - vrai si la suite de caractères introduite interactivement par l'utilisateur correspond à un composé chimique syntaxiquement correct alors :

COMP = l'identificateur du composé chimique identifié;

- faux sinon alors :

- message d'erreur.

2. Définition du langage

⟨ équation chimique ⟩ :: = ⟨ liste de produits ⟩⟨ type de réaction ⟩⟨ liste de produits ⟩

⟨ liste de produits ⟩ :: = ⟨ produit ⟩ / ⟨ produit ⟩ + ⟨ liste de produits ⟩

⟨ produit ⟩ :: = ⟨ coefficient stoechiométrique ⟩⟨ composé chimique ⟩

⟨ type de réaction ⟩ :: = ⟨ ⟺ ⟩ / ⟨ → ⟩

⟨ coefficient stoechiométrique ⟩ :: = ⟨ nombre exemplaire ⟩

⟨ composé chimique ⟩ :: = ⟨ composé ⟩⟨ charge ⟩

⟨ nombre exemplaire ⟩ :: = ⟨ vide ⟩ / ⟨ nombre* ⟩

⟨ nombre* ⟩ :: = ⟨ chiffre non nul ⟩ / ⟨ nombre* ⟩⟨ chiffre ⟩

⟨ chiffre non nul ⟩ :: = 1/2/3/4/5/6/7/8/9.

⟨ chiffre ⟩ :: = 0/1/2/3/4/5/6/7/8/9.

⟨ composé ⟩ :: = ⟨ composé sans ligand ⟩ / ⟨ composé avec ligand ⟩

⟨composé sans ligand⟩ :: = ⟨atome⟩⟨nombre exemplaire⟩ /
⟨atome⟩⟨nombre exemplaire⟩⟨composé sans ligand⟩
⟨composé avec ligand⟩ :: = ⟨composé sans ligand⟩ (⟨ligand⟩)
⟨nombre exemplaire⟩
⟨ligand⟩ :: = ⟨composé sans ligand⟩
⟨atome⟩ :: = H/He/Li/Be/.../U
⟨charge⟩ :: = ⟨vide⟩ /[⟨nombre exemplaire⟩⟨signe⟩]
⟨signe⟩ :: = +/-

3. Conception

a : la saisie

Parmi tous les caractères disponibles sur le clavier, nous avons distingué 3 sortes de caractères :

- les caractères "acceptables"
- les caractères "mémorisables"
- les caractères "significatifs"

Lors de la saisie d'un caractère au terminal, on n'acceptera le caractère en question que s'il fait partie de l'ensemble des caractères "acceptables" afin d'éviter qu'un utilisateur introduise des caractères qui n'interviennent jamais dans l'expression d'une équation ou d'un composé chimique. On diminuera ainsi le risque d'erreurs lors de l'analyse de la chaîne de caractères.

Parmi tous ces caractères "acceptables", on ne mémorisera que les caractères "mémorisables" dans le tableau LIGNE. On laissera toutefois la possibilité à l'utilisateur d'effacer des caractères déjà mémorisés.

La saisie terminée, on éliminera tous les caractères "blancs" pour ne plus qu'obtenir des caractères "significatifs".

Cependant, si l'utilisateur n'a introduit aucun caractère significatif ou plus de caractères significatifs que peut mémoriser le tableau LIGNE, un message d'erreur sera présenté à l'utilisateur.

b : analyse syntaxique

Nous avons associé à chaque forme syntaxique définie ci-dessus une fonction booléenne. Cette fonction sera vraie si on a pu retrouver dans la chaîne de caractères significatifs la forme syntaxique en question, fausse sinon.

Pour retrouver une forme syntaxique, on parcourera le tableau LIGNE de gauche à droite, poscc indiquant la position courante de la tête de lecture, en reconsituant progressivement la forme syntaxique en question.

En cas d'erreurs, la nature et l'emplacement de la première erreur rencontrée seront communiqués à l'utilisateur. Etant donné que la description d'une équation ou à fortiori d'un composé chimique dans le langage est assez brève, nous avons choisi de nous arrêter à la première erreur rencontrée.

c : analyse sémantique

Dans le cas où la chaîne de caractères représente une équation chimique, il faut vérifier si les équilibres électrique et stoechiométrique sont respectés.

Pour ce faire, il faut :

1. additionner la charge de chacun des réactifs (et des produits) pour obtenir la charge totale des réactifs (et des produits);

2. vérifier l'égalité entre la charge totale des réactifs et des produits;

3. comptabiliser le nombre d'atomes d'un type donné pour les réactifs (et les produits); ceci pour chaque type d'atome;

4. vérifier l'égalité entre le nombre d'atomes d'un type donné pour les réactifs et celui des produits; ceci pour chaque type d'atome.

Pour pouvoir comptabiliser le nombre d'atomes d'un type donné, nous allons définir la structure de données suivante :

type TABAT = array [ATOME] of NBATYDO
avec type ATOME = (H, He, Li, Be, ...,U)
type NBATYDO = entier représentant le nombre d'atomes du type en question.

Nous avons défini 3 primitives sur cette structure particulière :

- ADD (A,B) où A, B sont de type TABAT
  pour tout at ϵ atome :

$$A[At] \leftarrow A[At] + B[At].$$

- MUL (A, CS) où A est de type TABAT

  CS est " entier

  pour tout at ∈ atome :

  A [At] ← A [At] * CS.

- MIZERO (A) où A est de type TABAT

  pour tout at ∈ ATOME :

  A [At] ← 0 .

N.B.

Toutefois , les points 1 et 3 seront réalisés lors de l'analyse syntaxique afin d'éviter de relire une seconde fois la chaîne de caractères.

4. Algorithmes

```
FONCTION EQUACHIM (NC, NR, TCO, TCH, TYRE)
```

```
DEBUT
    EQUACHIM ← FALSE ; NC ← 0;
    SI LIRECHIMIE (LIGNE, LGLGN) ALORS
       SI SYNTAXIQUE (LIGNE, LGLGN, NC, NR, CHRE, CHPR,
                      TCO, TCS, TCH, TYRE, NBATRE, NBATPR)
                      ALORS
          SI SEMANTIQUE (CHRE, CHPR, NBATRE, NBATPR) ALORS
                EQUACHIM ← TRUE
FIN
```

```
FONCTION COMPCHIM (COMP)
```

```
DEBUT
    COMPCHIM ← FALSE;
    SI LIRECHIMIE (LIGNE, LGLGN) ALORS
       SI ANALCOMP (LIGNE, LGLGN, COMP) ALORS
          SI LONGUEUR (COMP) = LGLGN ALORS
                COMPCHIM ← TRUE
          SINON ERREUR (11);
FIN
```

PROCEDURE ADD (A, B)

```
DEBUT
    POUR TOUT AT ← H JUSQU'A  U FAIRE
            A [AT] ← A [AT] + B [AT] ;
FIN
```

PROCEDURE MULT (A, CS)

```
DEBUT
    POUR TOUT AT ← H JUSQU'A  U FAIRE
            A [AT] ← A [AT] * CS
FIN
```

PROCEDURE MIZERO (A)

```
DEBUT
    POUR TOUT AT ← H JUSQU'A  U FAIRE
            A [AT] ← 0;
FIN
```

FONCTION LIRECHIMIE (LIGNE, LGLGN)

Résultat : - VRAI si la suite de caractères introduite par l'élève contient au minimum un caractère significatif et au maximum LMLIGNE caractères significatifs alors

- pour tout $1 \leqslant i \leqslant$ LGLGN : LIGNE [i] = un caractère significatif.

- FAUX sinon alors

- message d'erreur.

```
DEBUT
    LIRECHIMIE ← TRUE;
    LGLGN ← 0;
    FIN ← FALSE;
    REPETER SAISIE (LIGNE, LGLGN)  JUSQU'A FIN;
    SI LGLGN = 0 ALORS   ERREUR (12);
                         LIRECHIMIE ← FALSE
    SINON  SI LGLGN >LMLIGNE ALORS   ERREUR (13);
                                     LIRECHIMIE ← FALSE
           SINON  ENLEVERBLANC (LIGNE, LGLGN);
                  SI LGLGN = 0  ALORS   ERREUR (14);
                                        LIRECHIMIE ← FALSE;
FIN
```

PROCEDURE SAISIE (LIGNE, LGLGN)

Résultat : - pour tout $1 \leqslant i \leqslant$ LGLGN : LIGNE [i] = un caractère mémorisable.

```
DEBUT
    REPETER LIRE (CLAVIER, CH) JUSQU'A ACCEPTABLE (CH);
    SI MEMORISABLE (CH) ALORS
                  LGLGN ← LGLGN + 1;   }  (* MEMORISATION)
                  LIGNE [LGLGN] ← CH;  }
                  ECRIRE (CH);         }
                  SI LGLGN ⩾ LMLIGNE ALORS FIN ← TRUE
```

```
        SINON
              SI  CH = 'ESC' ALORS FIN ← TRUE;
              SI  CH = '←' ALORS LIGNE [LGLGN] ← '' ;
                                 LGLGN ← LGLGN - 1;
                                 (* EFFACER LE DERNIER CARAC-
                                    TERE INSCRIT A L'ECRAN*)
FIN
```

PROCEDURE ENLEVERBLANC (LIGNE, LGLGN)

Argument : - pour tout $1 \leqslant i \leqslant$ LGLGN : LIGNE [i] = un caractère mémorisable.

Résultat : - pour tout $1 \leqslant i \leqslant$ LGLGN: LIGNE [i] = un caractère significatif.

```
DEBUT
        K ← 0; (*NOMBRE COURANT DE CARACTERES ≠''
                 RENCONTRES*)
        POUR I ← 1  jusqu'à LGLGN faire
              SI LIGNE [I] ≠ ' ' ALORS
                 K ← K + 1;
                 SI K < I ALORS
                    LIGNE [K] ← LIGNE [i] ;
                    LIGNE [i] ← ' ';
        LGLGN ← K; (NOMBRE DE CARACTERES SIGNIFICATIFS)
FIN
```

FONCTION SYNTAXIQUE (LIGNE, LGLGN, NC, NR, CHRE, CHPR, TCO, TCS, TYRE, NBATRE, NBATPR)

Argument : - pour tout $1 \leqslant i \leqslant$ LGLGN : LIGNE [i] = un caractère significatif.

Précondition : - NC = 0 (*nombre courant de composés identifiés*)

Résultat : - VRAI si découverte d'une suite de caractères dans LIGNE représentant une équation chimique.

<équation> :: = <liste de produits> < type de réaction > < liste de produits>

alors

- NC : le nombre de composés.
- NR : le nombre de réactifs.
- CHRE : la charge électrique totale des réactifs.
- CHPR : la charge électrique totale des produits.

pour tout $1 \leqslant i \leqslant$ NC :

- TCO [i] : l'identificateur du ième composé identifié.
- TCS [i] : le coefficient stoechiométrique du ième composé.
- TCH [i] : la charge électrique du ième composé.

pour tout $H \leqslant at \leqslant U$:

- NBATRE [At] : le nombre total d'atomes at pour les réactifs.

- NBATPR [At] : le nombre total d'atomes at pour les produits.

Postcondition : - pour tout $1 \leq i \leq NR$: TCO [i] = un réactif.
- pour tout $NR + 1 \leq i \leq NC$: TCO [i] = un produit.
- $NC > NR$.
- CC = '#', POSCC = lglgn, TYPECC = teol;

- FAUX sinon alors

- NUMERR : numéro du message d'erreur associé à la première erreur syntaxique rencontrée.
- POSCC : position où la première erreur syntaxique a été détectée dans le tableau LIGNE lors de la recherche d'une équation.

```
DEBUT
    SYNTAXIQUE ← FALSE;
    CC ← ' ';
    TYPE CC ← TBOL;
    POS CC ← 0 ;
    SI GETLISTEPRODUIT (NR, CHRE, NBATRE) ALORS
       SI GETYRE (TYRE) ALORS
             SI GETLISTEPRODUIT (NP, CHPR, NBATPR) ALORS
                          SYNTAXIQUE ← TRUE
       SINON ERREUR (10)
FIN
```

FONCTION GETLISTEPRODUIT (NCLP, CHLP, NBATLP)

Précondition : - POSCC : position courante à partir de laquelle il faut rechercher une liste de produits dans LIGNE.

<liste de produits> :: = <produit> / <produit> + <liste de produits>

Résultat : - VRAI si découverte d'une suite de caractères dans LIGNE représentant une liste de produits alors

- NCLP= nombre de composés identifiés dans la liste de produits.
- CHLP = charge électrique totale des composés identifiés dans la liste des produits.
- pour tout H ≤ at ≤ U :
  NBATLP[AT] = le nombre total d'atomes at pour les composés identifiés dans la liste de produits.

Postcondition : - NC = NC + NCLP.

- FAUX sinon alors

- POSCC = position où l'on a détecté la première erreur syntaxique dans LIGNE lors de la recherche d'une liste de produits.

NUMERR = numéro du message d'erreur correspondant .

```
DEBUT
    GETLISTEPRODUIT ← FALSE;
    NCLP ← 0;
    CHLP ← 0;
    MIZERO   (NBATLP);
    CONTINUE ← TRUE;
    REPETER
              NC ← NC + 1;
              SI GETPRODUIT (TCO [NC] , TCS [NC] , TCH [NC] ,
                             NBATPRO) ALORS
                        NCLP ← NCLP + 1;
                        CHLP ← CHLP + TCH [NC] ;
                        MUL (NBATPRO, TCS [NC] );
                        ADD (NBATLP, NBATPRO);
                        GETLISTEPRODUIT ← TRUE
              SINON     CONTINUE ← FALSE;
                        GETLISTEPRODUIT ← FALSE
     JUSQU'A CE QUE (PAS CONTINUE) OU CC = '+';
FIN
```

FONCTION GETYRE (TYRE)

Précondition : - POSCC : position courante à partir de laquelle il faut rechercher un type de réaction dans LIGNE.

⟨type réaction⟩ ::= ⟨⇔⟩ | ⟨→⟩

Résultat : - VRAI si découverte d'une suite de caractères dans LIGNE représentant un type de réaction alors

- TYRE : le type de réaction identifié.

- FAUX sinon alors

- POSCC : position dans LIGNE où la première erreur dans la recherche d'un type de réaction a été détectée.
- NUMERR : numéro du message d'erreur correspondant à cette erreur.

```
DEBUT
    GETYRE ← FALSE;
    SI CC='-' ALORS
        SI GETNEXT (CC, TYPECC) AND (CC = '-') ALORS
         SI GETNEXT (CC, TYPECC) AND (CC = '-') ALORS
                    TYRE ← DIRECTE;
                    GETYRE ← TRUE
    SINON SI CC = '<' ALORS
           SI GETNEXT (CC, TYPECC) AND (CC ='=') ALORS
            SI GETNEXT (CC, TYPECC) AND (CC ='=') ALORS
             SI GETNEXT (CC, TYPECC) AND (CC ='>') ALORS
                        TYRE ← EQUILIBREE;
                        GETYRE ← TRUE
FIN
```

FONCTION GETPRODUIT (CO, CS, CH, NBATPR)

Précondition : - POSCC = position courante à partir de laquelle il faut rechercher un produit dans LIGNE.

⟨produit⟩ ::= ⟨coefficient stoechiométrique⟩ ⟨composé⟩ ⟨charge⟩

Résultat : - VRAI si découverte d'une suite de caractères dans LIGNE représentant un produit alors

- CO = l'identificateur du composé identifié.
- CS = le coefficient stoechiométrique du composé identifié.
- CH = la charge électrique du composé identifié.

pour tout H ⩽ at ⩽ U :

- NBATPR [AT] = le nombre d'atomes at identifié dans le produit .

Postcondition : - NOT EXISTDEJA (NC).

- FAUX sinon alors

- POSCC : position où l'on a détecté la première erreur syntaxique dans LIGNE lors de la recherche d'un produit.
- NUMERR : numéro du message d'erreur correspondant.

```
DEBUT
    GETPRODUIT ← FALSE;
    SI GETNEXT (CC, TYPECC) ALORS
     SI PAS GETMULT (CS) ALORS ERREUR (1) SINON
        SI GETCOMPOSE (CO, NBATPR) ALORS
           SI GETCHARGE (CH) ALORS
              CONCATENA (CO, CH);
              SI EXISTDEJA (NC) ALORS ERREUR (7)
              SINON GETPRODUIT ← TRUE;
FIN
```

FONCTION GETNEXT (CC, TYPECC)

Précondition : - POSCC : position courante à partir de laquelle il faut lire le prochain caractère dans LIGNE.

Résultat et postconditions :

- VRAI si POSCC < LGLGN alors
  - POSCC = POSCC + 1
  - CC : le nouveau caractère lu .
  - TYPECC : le type du caractère lu.
- FAUX sinon alors
  - POSCC = LGLGN.
  - CC = '#'.
  - TYPECC = TEOL.

```
DEBUT
    SI POSCC ≥ LGLGN ALORS
        CC ← '#';
        TYPECC ← TEOL
    SINON
        POSCC ← POSCC + 1;
        CC ← LIGNE [POSCC ];
        TYPECC ← TYPECH (CC);
  GETNEXT ← CC ≠ '#';
FIN
```

FONCTION GETMULT (MULT)

Précondition : - POSCC : position courante à partir de laquelle il faut rechercher un nombre exemplaire dans LIGNE.

⟨nombre exemplaire⟩ ::= ⟨vide⟩/⟨nombre⟩.

⟨nombre*⟩ ::=⟨chiffre non nul⟩ !⟨nombre⟩*⟨chiffre⟩ .

⟨chiffre non nul⟩ ::= 1!2!3!4!5!6!7!8!9

⟨chiffre⟩ ::=0!1!2!3!4!5!6!7!8!9

Résultat : - VRAI si découverte d'une suite de caractères dans LIGNE représentant un nombre exemplaire alors

- MULT : la valeur du nombre exemplaire identifié.

- FAUX sinon alors

- POSCC : position dans LIGNE où la

première erreur a été détectée.

- NUMERR : numéro du message d'erreur correspondant à cette erreur.

```
DEBUT
    SI CC = 'O' ALORS
            MULT ← Ø ;
            GETMULT ← FALSE
    SINON
        SI TYPECC = NUM ALORS
                MULT ← VAL (CC);
                TANT QUE GETNEXT (CC, TYPECC) AND
                ((TYPECC = NUM) OR (CC = Ø)) FAIRE
                  MULT ← 10 * MULT + VAL(CC);
                  GETMULT ← TRUE
        SINON   MULT ← 1;
                GETMULT ← TRUE;
FIN
```

FONCTION GETCOMPOSE (COMPOSE, NBATCO)

Précondition : - POSCC : position courante à partir de laquelle il faut rechercher un composé dans LIGNE.

⟨composé⟩ ::= ⟨composé sans ligand⟩ | ⟨composé avec ligand⟩

⟨composé sans ligand⟩ ::= ⟨atome⟩⟨nombre exemplaire⟩ |
⟨atome⟩⟨nombre exemplaire⟩
⟨composé sans ligand⟩

⟨composé avec ligand⟩::⟨composé sans ligand⟩ (⟨ligand⟩)
⟨nombre exemplaire⟩

Résultat : VRAI si découverte d'une suite de caractères dans LIGNE représentant un composé alors
- COMPOSE : l'identificateur du composé identifié.
- pour tout H ⟨=at ⟨=U : NBATCO [at] = nombre d'atomes at pour le composé identifié.

FAUX sinon alors
- POSCC : position dans LIGNE de la première erreur syntaxique détectée dans la recherche d'un composé.
- NUMERR : numéro du message d'erreur correspondant à cette erreur.

```
DEBUT
    GETCOMPOSE ← FALSE;
    COMPOSE ← ' ';
    MIZERO (NBATCO);
    TANT QUE GETATOME (SIMPLE, ATOM) ET GETMULT (MULT) FAIRE
            NBATCO [ATOM] ← NBATCO [ATOM] + MULT;
            COMPOSE ← CONCAT (COMPOSE, SIMPLE, CAR(MULT));
    SI LONGUEUR(COMPOSE)≠0 ET MULT ≠ 0 ALORS
       SI CC='('ALORS
          SI GETMULT (CC, TYPECC) ET GETLIGAND (LIGAND,NBATLI)
                  ET GETMULT (MULT) ALORS
```

```
            COMPOSE ← CONCAT (COMPOSE,'(', LIGAND,')',
                              CAR (MULT));
            MUL (NBATLI,MULT) ;
            ADD (NBATCO,NBATLI) ;
            GETCOMPOSE ← TRUE
        SINON GETCOMPOSE ← TRUE ;
```

FIN

FONCTION GETLIGAND (LIGAND, NBATLI)

Précondition: - POSCC : position courante à partir de laquelle il faut chercher un ligand dans LIGNE.

<ligand> ::=<composé sans ligand>

Résultat : - VRAI si découverte d'une suite de caractères dans LIGNE représentant un ligand alors
- LIGAND : l'identificateur du ligand identifié.
- pour tout H<=at<=U : NBATLI [at] = nombre d'atomes at pour le ligand identifié

-FAUX sinon alors
- POSCC : position dans LIGNE où la première erreur dans la recherche d'un ligand a été détectée.
- NUMERR : numéro du message d'erreur correspondant à cette erreur.

```
DEBUT
    GETLIGAND ← FALSE;
    LIGAND ← ' ';
    MIZERO (NBATLI);
      TANT QUE GETATOME (SIMPLE, ATOM) AND GETMULT (MULT)
      FAIRE
           NBATLI [ATOM] ← NBATLI [ATOM] + MULT;
           LIGAND ← CONCAT (LIGAND, SIMPLE, CAR (MULT));
    SI LONGUEUR (LIGAND) ≠ 0 AND MULT ≠ 0 ALORS
           SI CC = ')' ALORS
                    B ← GETNEXT (CC, TYPECC);
                    GETLIGAND ← TRUE
           SINON ERREUR (3);
FIN
```

FONCTION GETATOME (SIMPLE, ATOM)

Précondition : - POSCC : position courante à partir de laquelle il faut rechercher un atome dans LIGNE.

<atome> ::=H|He|Li|Be|B|C|N|O|F|...|U

Résultat : - VRAI si découverte d'une suite de caractères dans LIGNE représentant un atome alors

- SIMPLE:l'identificateur de l'atome identifié.
- ATOM : l'atome identifié (type ATOME).

- FAUX sinon alors

-POSCC : position dans LIGNE où la première erreur dans la recherche d'un atome a été détectée.

- NUMERR : numéro du message d'erreur correspondant à cette erreur.

```
DEBUT
    GETATOME ← FALSE;
    SIMPLE ← ' ' ;
    SI TYPECC = MAJ ALORS
       SIMPLE [1] ← CC;
       COMPT ← 1;
       TANT QUE GETNEXT (CC, TYPECC) AND (TYPECC = MIN)
       FAIRE
            COMPT ← COMPT + 1;
            SIMPLE [COMPT] ← CC;
    SI SIMPLE ≠ ' ' ALORS
       SI EXISTATOME (SIMPLE, ATOM) ALORS
                  GETATOME ← TRUE
       SINON ERREUR (2);
FIN
```

FONCTION EXISTATOME (SIMPLE, ATOM)

Argument : - SIMPLE : un identificateur.

Résultat : - VRAI si SIMPLE est bien un atome alors

- ATOM : l'atome en question.

- FAUX sinon.

```
DEBUT
    EXISTATOME ← FALSE;
    ATOM ← CONV (SIMPLE);
    POUR AT ← H JUSQU'A U FAIRE
        SI ATOM = AT ALORS EXISTATOME ← TRUE;
FIN
```

FONCTION GETCHARGE (CHARGE)

Précondition : - POSCC : position courante à partir de laquelle il faut rechercher une une charge dans LIGNE.

<charge> ::= <nombre exemplaire> < signe>

Résultat : - VRAI si découverte d'une suite de caractères dans LIGNE représentant une charge alors

- CHARGE : la valeur de la charge identifiée.

- FAUX sinon alors

- POSCC : position dans LIGNE où la première erreur dans la recherche d'une charge a été détectée.
- NUMERR : numéro du message d'erreur correspondant à cette erreur.

```
DEBUT
    GETCHARGE ← TRUE;
    CHARGE ← 0;
    SI CC = '[ ' ALORS
       SI GETNEXT (CC, TYPECC) AND GETMULT (MULT) ALORS
          SI GETSIGNE (SIGNE) ALORS
             SI SIGNE = '+' ALORS CHARGE ← MULT
             SINON CHARGE ← -MULT;
             SI GETNEXT (CC, TYPECC) AND CC =' ] 'ALORS
                B ← GETNEXT (CC, TYPECC);
                GETCHARGE ← TRUE
             SINON  GETCHARGE ← FALSE;
                    ERREUR (6)
          SINON GETCHARGE ← FALSE;
                ERREUR (5)
       SINON GETCHARGE ← FALSE
             ERREUR (4)
FIN
```

FONCTION GETSIGNE (SIGNE)

Précondition : - POSCC : position courante à partir de laquelle il faut rechercher un signe dans LIGNE.

<signe> ::=+!-

Résultat : - VRAI si découverte d'une suite de caractères dans LIGNE représentant un signe alors

- SIGNE : le type du signe identifié.

- FAUX sinon alors

- POSCC : position dans LIGNE où la première erreur dans la recherche d'un signe a été détectée.
- NUMERR : numéro du message d'erreur correspondant à cette erreur.

```
DEBUT
    GETSIGNE ← TRUE;
    SI CC = ' +' THEN SIGNE ← '+'
    SINON SI CC = '-'  ALORS SIGNE ← '-'
          SINON GETSIGNE ← FALSE;
FIN
```

FONCTION EXISTDEJA (N)

Argument : - N = numéro du composé.

Précondition : - $1 \leqslant N \leqslant NC$.

Résultat : - VRAI si pour tout $1 \leqslant i \leqslant NC$ et $i \neq N$: $TCO[i] \neq TCO[N]$.

- FAUX sinon.

```
DEBUT
    EXISTDEJA ← FALSE
    TANT QUE I ⩽ (N-1) FAIRE
         SI TCO[I] = TCO[N] ALORS
              EXISTDEJA ← TRUE
         I ← I + 1
FIN
```

FONCTION ANALCOMP (LIGNE, LGLGN, COMP)

Argument : - pour tout $1 \leq i \leq$ LGLGN : LIGNE [i] = un caractère significatif.

Résultat : - VRAI si découverte dans le tableau LIGNE d'une suite de caractères représentant un composé chimique alors

- COMP = le composé chimique identifié.

Postcondition : - POSCC = LGLGN : CC = '#'; TYPECC = TEOL.

- FAUX sinon alors

- POSCC = position où l'on a détecté la première erreur syntaxique dans LIGNE lors de la recherche d'un composé chimique.
- NUMERR = numéro du message d'erreur correspondant.

```
DEBUT
    CC ← ' ';
    TYPECC ← TBOL;
    POSCC ← 0;
    ANALCOMP ← FALSE;
    SI GETNEXT (CC, TYPECC) AND GETCOMPOSE (COMP, NBATCO)
       AND GETCHARGE (CHARGE) ALORS
           ANALCOMP ← TRUE;
           CONCATENA (COMP, CHARGE);
FIN
```

FONCTION SEMANTIQUE (CHRE, CHPR, NBATRE, NBATPR)

Arguments : - CHRE : la charge électrique totale des réactifs.
- CHPR : la charge électrique totale des produits.

Pour tout H ≤ at ≤ U :

- NBATRE [at] : le nombre total d'atomes at pour les réactifs.
- NBATPR [at] : le nombre total d'atomes at pour les produits.

Résultat : - VRAI si l'équation est sémantiquement correcte càd :

- l'équilibre électrique est respecté.
- l'équilibre stoechiométrique est respecté.

- FAUX sinon alors

- message d'erreur si la réaction n'est pas électriquement équilibrée.
- message d'erreur si la réaction n'est pas stoechiométriquement équilibrée.

```
DEBUT
    SEMANTIQUE ← FALSE;
    SI EQUILELEC (CHRE, CHPR) ALORS
       SI EQUILSTOECHIO (NBATRE, NBATPR) ALORS
```

SEMANTIQUE ← TRUE

SINON ERREUR (8)

SINON ERREUR (9)

FIN

FONCTION EQUILELEC (CHRE, CHPR)

Arguments : - CHRE : la charge électrique totale des réactifs.

- CHPR : la charge électrique totale des produits.

Résultats : - VRAI si la réaction est électriquement équilibrée càd CHRE = CHPR.

- FAUX sinon.

DEBUT

EQUILELEC ← TRUE;

SI CHRE ≠ CHPR ALORS EQUILELEC ← FALSE;

FIN

FONCTION EQUILSTOECHIO (NBATRE, NBATPR)

Arguments : Pour tout H <=at <=U :

- NBATRE [at] : le nombre total d'atomes at pour les réactifs.
- NBATPR [at] : le nombre total d'atomes at pour les produits.

Résultat : - VRAI si la réaction est stoechiométriquement équilibrée càd
si pour tout H ≤ at ≤ U : NBATRE [at] = NBATPR [at]

- FAUX sinon.

```
DEBUT
    EQUILSTOECHIO ← TRUE;
    POUR TOUT AT ← H JUSQU'A U FAIRE
            SI NBATRE [AT] ≠ NBATPR [AT] ALORS
              EQUILSTOECHIO ← FALSE;
FIN
```

PROCEDURE CONCATENA (COMP, CHARGE)

Arguments : - COMP : un composé.
- CHARGE : charge du composé.

Résultat : - COMP = CONCAT (COMP, '[', CHARGE, ']').

```
DEBUT
    SI CHARGE > 0 ALORS COMP ← CONCAT (COMP, '[ ', CAR( ABS
    (CHARGE)), ! +]');
    SI CHARGE < 0 ALORS COMP← CONCAT (COMP, '[ ', CAR (ABS
    (CHARGE)), '-]');
FIN
```

PROCEDURE ERREUR (NUM)

```
DEBUT
    ECRIRE ('MESSAGE D'ERREUR', NUM, 'AU CARACTERE', POSCC)
FIN
```

B Module "VALEURS STANDARDS"

1. Définition de l'entité d'information

```
ARTICLE = RECORD
                IDENTIFIANT;
                ATT : ATTRIBUT;
          END
IDENTIFIANT = COMPCHIM (*selon le langage défini*)
ATTRIBUT = RECORD
                HOF : REAL;
                SOF : REAL;
           END
```

2. Conception

Les différentes entités d'information seront enregistrées sur un fichier séquentiel vu le nombre peu important de modifications à apporter aux entités en question. Toutefois, nous avons envisagé de disposer en mémoire centrale des entités d'information relatives aux composés chimiques les plus fréquement utilisés afin de réduire le temps d'accès à une information.

Pour ce faire, les composés chimiques les plus fréquement utilisés seront également enregistrés sur un fichier séquentiel. Cependant, dès le début d'exécution du programme, on procédera au chargement de ce fichier en mémoire centrale (dans un tableau permettant l'accès direct à l'information recherchée).

Ainsi lors d'un accès à une entité d'information, on testera tout d'abord son existence en mémoire centrale

puis, seulement si elle n'y existe pas, en mémoire secondaire.

Cette organisation des données nécessite cependant quelques précautions :

- lors de l'enregistrement d'une nouvelle entité d'information, on devra veiller à son unicité. En effet, on ne peut dans aucun cas avoir
  - une même entité reprise en mémoire secondaire et en mémoire centrale.
  - une même entité reprise deux fois en mémoire centrale ou en mémoire secondaire.
- il faudra également veiller à ce que le fichier séquentiel contenant les composés chimiques les plus fréquement utilisés puisse être chargé complètement en mémoire centrale. Lors d'un enregistrement d'une nouvelle entité dans ce fichier, on devra vérifier que le nombre d'entités mémorisées dans le fichier soit strictement inférieur à la longueur maximale du tableau en mémoire centrale.

3. Fonctions de l'interface

Soit fich-comp = fichier à accès séquentiel d'ARTICLE

tabl-comp = tableau [1...LMTABL] d'ARTICLE

a: fonction existmolecule

Arguments : - F : fich-comp.
- T : tabl-comp.
- I : identifiant.
- N : nombre de composés chimiques enregistrés dans T.

Résultat : - VRAI si le composé chimique identifié par I est repris dans le fichier séquentiel F ou dans le tableau T alors :
- ATT. HO : enthalpie libre standard de formation du composé I.
- ATT. SO : entropie absolue du composé I.

- FAUX sinon.

b : fonction existmc

Arguments : - T : tabl-comp.
- I : identifiant.
- N : nombre de composés chimiques enregistrés dans T.

Résultat : - VRAI si le composé chimique identifié par I est repris dans le tableau T alors
- ATT. HO : enthalpie libre standard de formation du composé I.
- ATT. SO : entropie absolue du composé I.

- FAUX sinon.

c : fonction existms

Arguments : - F : fich-comp.

- I : identifiant.

Résultat : - VRAI si le composé chimique identifié par I est repris dans le fichier séquentiel F alors

- ATT.HO : enthalpie libre standard de formation du composé I.
- ATT.SO : entropie absolue du composé I.

- FAUX sinon.

d: procédure enregis

Arguments : - F : fich-comp.

- A : article.

Précondition : not existms (F, I, ATT) tel que I appartient à A = (I, ATT).

Résultat : - F' : fich-comp.

Postcondition : $F' = F \cup A$.

existms (F, I, ATT) tel que I appartient à A = (I, ATT).

e : procédure modifie

Arguments : - F : fich-comp.

- A : article tel que A = (I, ATT).

Précondition : existms (F, I, ATT) tel que I appartient à A' = (I, ATT').

Résultat : - F' : fich-comp.

Postcondition : existms (F', I, ATT) tel que I appartient à A = (I, ATT).
et not existms (F', I, ATT') tel que I appartient à A' = (I, ATT').

f : procédure suppres

Arguments : - F : fich-comp.
- I : identifiant.

Précondition : existms (F, I, ATT) tel que I appartient à A = (I, ATT).

Résultat : - F' : fich-comp.

Postcondition : F' = F \ A tel que A = (I, ATT).
not existms (F', I, ATT) tel que I appartient à A = (I, ATT).

g:procédure consfseq

Argument : - F : fich-comp.

Résultat : - si F est vide alors message "fichier vide"
sinon ATT tel que ATT appartient à A = (I, ATT)
pour tout A appartenant à F.

h : procédure consid

Argument : - F : fich-comp.
- I : identifiant.

Résultat : - si existms (F, I, ATT) alors

ATT tel que ATT appartient à A = (I,ATT)

sinon message d'inexistence .

i : fonction compter

Argument : - F : fich-comp .

Résultat : - le nombre de composés chimiques mémorisés dans le fichier séquentiel F .

j : procédure charger

Arguments : - F : fich-comp .

- T : tabl-comp .

Précondition : compter (F) $<$ LMTABL .

Résultat : pour tout $1 \leqslant i, j \leqslant$ compter (F) :

si $i < j$ alors $T[i]$ = A appartenant à F

$T[j] = A'$ " à F

avec A précédant A' dans le fichier F.

k : procédure tri

Argument : - F : fich-comp .

- N : compter (F) .

Résultat : - F' : fich comp .

Postcondition : F' = F trié sur I .

4. Algorithmes

FONCTION EXISTMOLECULE (F, T, N, I, ATT)

```
DEBUT
    EXISTMOLECULE ← TRUE;
    SI NOT EXISTMC (T, N, I, ATT) ALORS
       SI NOT EXISTMS (F, I, ATT) ALORS
              EXISTMOLECULE ← FALSE
FIN
```

FONCTION EXISTMS (F, I, ATT)

```
DEBUT
    TROUVE ← FALSE;
    OUVRIR - LECTURE (F);
    TANT QUE PAS FIN-FICHIER (F) ET PAS TROUVE FAIRE
             SI I = (ID)F ALORS
                  TROUVE ← TRUE;
                  ATT. HOF ← (AT. HOF)F;
                  ATT. SOF ← (AT. SOF)F
             SINON SUIVANT (F)
    EXISTMS ← TROUVE;
FIN
```

```
FONCTION EXISTMC (T, N, I, ATT)

DEBUT
    TROUVE ← FALSE;
    J ← 1;
    TANT QUE J ≤ N ET PAS TROUVE FAIRE
        SI I = (ID)_T ALORS
                TROUVE ← TRUE;
                ATT.HOF ← (AT.HOF)_T;
                ATT.SOF ← (AT.SOF)_T
        SINON J ← J + 1 ;
    EXISTMC ← TROUVE
FIN
```

```
PROCEDURE ENREGIS (F, A)

DEBUT
    OUVRIR - LECTURE(F);
    OUVRIR - ECRITURE(AF);
    TANT QUE PAS FIN-FICHIER (F) FAIRE
            COPIER (F → AF);
            SUIVANT (F);
    FERMER (F);
    ECRIRE (A,AF);
    FERMER (AF);
    TRANSFERER (AF, F);
FIN
```

PROCEDURE MODIFIE (F, A)

```
DEBUT
    OUVRIR LECTURE (F);
    OUVRIR-ECRITURE (AF);
    TANT QUE PAS FIN-FICHIER (F) FAIRE
            SI A.I ≠ (ID)F ALORS
                COPIER (F → AF);
                SUIVANT (F)
            SINON ECRIRE (A,AF);
                  SUIVANT (F) ;
    FERMER (F);
    FERMER (AF);
    TRANSFERER (AF, F);
FIN
```

PROCEDURE SUPPRES (F, I)

```
DEBUT
    OUVRIR - LECTURE (F);
    OUVRIR - ECRITURE (AF);
    TANT QUE PAS FIN - FICHIER (F) FAIRE
            SI I = (ID)F ALORS
                   COPIER (F→ AF);
                   SUIVANT (F)
            SINON SUIVANT (F)
    FERMER (F);
    FERMER (AF);
    TRANSFERER (AF, F);
FIN
```

```
PROCEDURE CONFSEQ (F)

DEBUT
    OUVRIR - LECTURE (F);
    SI FIN - FICHIER (F) ALORS MESSAGE
    SINON
       TANT QUE PAS FIN - FICHIER (F) FAIRE
              ECRIRE (ECRAN,(A.ATT)F);
              SUIVANT (F);
    FERMER (F);
FIN
```

```
PROCEDURE CONSID (F, I)

DEBUT
    SI EXISTMS (F, I, ATT) ALORS
          ECRIRE (ECRAN, ATT)
    SINON MESSAGE;
FIN
```

```
FONCTION COMPTER (F)

DEBUT
    OUVRIR - LECTURE (F);
    N ← 0;
    TANT QUE PAS FIN - FICHIER (F) FAIRE
          N ← N + 1;
          SUIVANT (F);
    FERMER (F);
    COMPTER ← N;
FIN
```

PROCEDURE CHARGER (F, T)

```
DEBUT
    OUVRIR - LECTURE (F);
    N ← 0;
    TANT QUE PAS FIN - FICHIER (F) FAIRE
            N ← N + 1 ;
            T [N] ← (A)F;
            SUIVANT (F);
    FERMER (F);
FIN
```

PROCEDURE TRANSFERER (AF, F)

```
DEBUT
    OUVRIR - LECTURE (AF);
    OUVRIR - ECRITURE ( F);
    TANT QUE PAS FIN - FICHIER (AF) FAIRE
            COPIER (AF → F);
            SUIVANT (AF);
    FERMER (AF);
    FERMER (F)
FIN
```

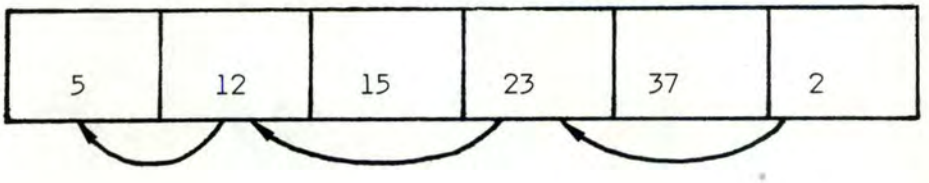
PROCEDURE TRI (F, NBAT)

```
DEBUT
    LECTURE;
    TRISHELL (1, NBAT);
    ECRITURE ;
FIN
```

Méthode du tri - SHELL

Dans la méthode du tri par insertion , chaque élément est comparé à ses voisins de gauche les uns après les autres.

sauts de 1 place : 5 tests pour placer la valeur 2 .



sauts de 2 places : 3 tests pour placer la valeur 2 .

Dans la méthode du tri.shell, on effectura des sauts importants dans le tableau pour placer plus rapidement les éléments.

On définit une grandeur de saut, soit pas, et on considère tous les sous-ensembles d'indices

1, 1 + pas, 1 + 2 pas, 1 + 3 pas, ...

2 + 2 + pas + 2 + 2 pas, ...

⋮

pas, 2 pas, 3 pas, ...

On ordonne chacun de ces sous-ensembles, considérés comme des tableaux, par insertion.

Après avoir appliqué la procédure de tri par insertion à tous les sous-ensembles construits à partir d'un pas donné, les places d'indices (1, 2, 3, ...pas) contiennent les minima de ces sous-ensembles.

On recommence la procédure avec un autre pas inférieur au premier, et ainsi de suite jusqu'à un pas égal à 1 .

Pour cette dernière valeur, pas = 1, nous faisons en fait un tri par insertion simple avec la différence que le tableau est déjà partiellement ordonné.

Procédure tri-shell (gauche, droite)

```
DEBUT
    PAS ← 1;
    TANT QUE PAS < (DROITE - GAUCHE + 1)/9 FAIRE
            PAS ← 3* PAS + 1; (*RECHERCHE DE LA VALEUR DU
                                PAS*)
    REPETER
```

```
        POUR K ← GAUCHE JUSQU'A PAS FAIRE
            (*TRI PAR INSERTION DU TABLEAU
                    K, K + PAS, K + 2 PAS,... *)
          POUR I ← K + PAS  JUSQU'A DROITE
              S ← T[I] ;
              J ←I-PAS;
              TANT QUE J ⩾ K + PAS ET CLEF (J) > CLEF(S)
              FAIRE
                  T [J + PAS] ← T [J] ;
                  J ← J - PAS;
              SI CLEF (K) > CLEF (S) ALORS
                  J ← K-PAS;
                  T [K+PAS] ←  T [K] ;
              T [J] ← S;
        PAS ← PAS/3
  JUSQU'A  PAS = 0
FIN
```

C. Module "FONCTIONS CHIMIQUES"

a. Fonctions nécessitant l'accès aux informations mémorisées

a.1. Conception

L'évaluation de chacune des fonctions chimiques nécessaires notamment au calcul de la valeur de la constante d'équilibre a largement été décrite lors de l'analyse fonctionnelle (chapitre III paragraphe 3).

Nous avions alors attiré l'attention sur le fait que cette évaluation pouvait ne pas toujours aboutir. C'est pourquoi, ces diverses fonctions ont été implémentées sous forme de fonctions booléennes. Chacune de celles-ci sera vraie si on a pu déterminer la valeur de la fonction chimique correspondante, auquel cas la valeur sera disponible, fausse sinon.

Rappelons que, lors de cette évaluation, on recherchera tout d'abord les informations en mémoire centrale puis, si elles ne sont pas présentes, en mémoire secondaire. Ceci suppose bien entendu qu'on ait chargé, dès le début d'exécution du module, les informations concernant les composés les plus utilisés en mémoire centrale.

a.2. Fonctions de l'interface

FONCTION GETKE

Arguments : - NR : le nombre de réactifs que comporte la réaction.

- NC : le nombre de composés que comporte la réaction.
- T : la température à laquelle la réaction a lieu.

pour tout $1 \leq i \leq NC$:

- TCO [i] : l'identificateur du ième composé.
- TCS [i] : le coefficient stoechiométrique du composé i.

Précondition : pour tout $1 \leq i \leq NR$: TCO [i] : un réactif.
pour tout $NR+1 \leq i \leq NC$: TCO [i] : un produit.
$NC > NR$.

Résultat : - VRAI si on a pu déterminer la constante d'équilibre de la réaction
alors - KE : la valeur de la constante d'équilibre de la réaction.

- FAUX sinon
alors messages d'erreur.

FONCTION GETDGO

Arguments : - NR : le nombre de réactifs que comporte la réaction.

- NC : le nombre de composés que comporte la réaction.
- T : la température à laquelle la réaction a lieu.

pour tout $1 \leq i \leq NC$:

- TCO [i] : l'identificateur du ième composé.
- TCS [i] : le coefficient stoechiométrique du composé i.

Précondition : pour tout $1 \leq i \leq NR$: TCO [i] : un réactif.
pour tout $NR+1 \leq i \leq NC$: TCO [i] : un produit.
$NC > NR$.

Résultat : - VRAI si on a pu déterminer l'énergie libre de la réaction
alors DGO : l'énergie libre de la réaction.

- FAUX sinon
alors messages d'erreur.

FONCTION GETDHO

Arguments : - NR : le nombre de réactifs que comporte la réaction.

- NC : le nombre de composés que comporte la réaction.

pour tout $1 \leq i \leq NC$:

- TCO [i] : l'identificateur du ième composé.
- TCS [i] : le coefficient stoechiométrique du composé i.

Précondition : pour tout $1 \leq i \leq NR$: TCO [i] : un réactif.
pour tout $NR+1 \leq i \leq NC$: TCO [i] : un produit.
$NC > NR$.

Résultat : - VRAI si on a pu déterminer l'enthalpie de la réaction
alors DHO : l'enthalpie libre de la réaction.

- FAUX sinon
alors messages d'erreur.

FONCTION GETDSO

Arguments : - NR : le nombre de réactifs que comporte la réaction.

- NC : le nombre de composés que comporte la réaction.

pour tout $1 \leq i \leq NC$:

- TCO [i] : l'identificateur du ième composé.
- TCS [i] : le coefficient stoechiométrique du composé i.

Précondition : pour tout $1 \leq i \leq NR$: TCO [i] :un réactif.
pour tout $NR+1 \leq i \leq NC$: TCO [i] :un produit.
$NC > NR$.

Résultat : - VRAI si on a pu déterminer l'entropie de la réaction
alors DSO : l'entropie de la réaction.

- FAUX sinon
alors messages d'erreur.

FONCTION GETDGP

Arguments : - NR : le nombre de réactifs que comporte la réaction.
- NC : le nombre de composés que comporte la réaction.
- T : la température à laquelle la réaction a lieu.

pour tout $NR+1 \leq i \leq NC$:

- TCO [i] : l'identificateur du ième produit.
- TCS [i] : le coefficient stoechiométrique du produit i.

Résultat : - VRAI si on a pu déterminer l'énergie libre des produits.
alors DGP : l'énergie libre des produits.

- FAUX sinon
alors messages d'erreur.

FONCTION GETDHP

Arguments : - NR : le nombre de réactifs que comporte la réaction.
- NC : le nombre de composés que comporte la réaction.
pour tout NR+1 $\leq$ i $\leq$ NC :
- TCO [i] : l'identificateur du ième produit.
- TCS [i] : le coefficient stoechiométrique du produit i.

Résultat : - VRAI si on a pu déterminer l'enthalpie des produits
alors DHP : l'enthalpie des produits.

- FAUX sinon
alors message d'erreur.

FONCTION GETDSP

Arguments : - NR : le nombre de réactifs que comporte la réaction.

- NC : le nombre de composés que comporte la réaction.

pour tout NR+1 ≤ i ≤ NC :

- TCO [i] : l'identificateur du ième produit.
- TCS [i] : le coefficient stoechiométrique du produit i.

Résultat : - VRAI si on a pu déterminer l'entropie des produits

alors DSP : l'entropie des produits.

- FAUX sinon

alors message d'erreur.

FONCTION GETDGR

Arguments : - NR : le nombre de réactifs que comporte la réaction

- T : la température à laquelle la réaction a lieu.

pour tout 1 ≤ i ≤ NR :

- TCO [i] : l'identificateur du ième réactif.
- TCS [i] : le coefficient stoechiométrique du réactif i.

Résultat : - VRAI si on a pu déterminer l'énergie libre des réactifs

alors DGP : l'énergie libre des réactifs.

- FAUX sinon.

alors messages d'erreur.

FONCTION GETDHR

Arguments : - NR : le nombre de réactifs que comporte la réaction.

pour tout $1 \leq i \leq NR$:

- TCO [i] : l'identificateur du ième réactif.
- TCS [i] : le coefficient stoechiométrique du réactif i.

Résultat : - VRAI si on a pu déterminer l'enthalpie des réactifs

alors DHR : l'enthalpie des réactifs.

- FAUX sinon

alors message d'erreur.

FONCTION GETDSR

Arguments : - NR : le nombre de réactifs que comporte la réaction.

pour tout $1 \leq i \leq NR$:

- TCO [i] : l'identificateur du ième réactif.
- TCS [i] : le coefficient stoechiométrique du réactif i.

Résultat : - VRAI si on a pu déterminer l'entropie des réactifs.
alors DSP : l'entropie des réactifs.

- FAUX sinon
alors message d'erreur.

---

a.3. Algorithmes

FONCTION GETDHR (TC, TCS, NR, DHR)

```
DEBUT
    GETDHR ← TRUE;
    CONTINUE ← TRUE;
    DHR ← 0;
    J ← 1;
    TANT QUE (J ≤ NR) ET CONTINUE FAIRE
          I ← TC [J] ;
          SI EXISTMOLECULE (F, T, NBCMC, I, ATT) ALORS
             DHR ← DHR + (ATT.HOF * TCS [ J] );
             J ← J+1
          SINON
             GETDHR ← FALSE;
             CONTINUE ← FALSE;
```

ERREUR;

FIN

Remarque :

NBCMC représente le nombre de composés chimiques qui ont été chargés en mémoire centrale à partir du fichier séquentiel contenant les composés chimiques les plus fréquement utilisés.

b. Fonctions ne nécessitant pas d'accès aux informations mémorisées

b.1. Conception

Parmi ces fonctions, on ne retiendra à notre attention que la fonction permettant d'obtenir des valeurs de concentrations à l'équilibre à partir des valeurs de concentrations initiales en réactif.

La façon de procéder a largement été décrite au chapître V paragraphe 2.

Le noeud du problème consiste donc à trouver une racine x_r du polynôme

$$F(x) = K * \text{produit}\left[k:1..nr\right]\ \left(\left[c_k\right]_i - cs_k * x\right)^{cs_k} - \text{produit}\left[j:nr+1..nc\right]\ \left(cs_j * x\right)^{cs_j} = 0.$$

sachant que $0 < x_r < \min\ \left(\left[c_j\right]_i / cs_j\right)$ pour tout $1 \leqslant j \leqslant nr$

La recherche de cette racine se fera par méthode dichotomique.

Soit x_r une racine séparée de l'équation $F(x) = 0$ dans l'intervalle $[x_a, x_b]$ tel que $F(x_a) * F(x_b) < 0$.

Désignons par x_0, le milieu de l'intervalle $[x_A, x_B]$. La racine x_r est dans l'un des intervalles $[x_A, x_0]$ ou $[x_0, x_B]$. Pour savoir lequel des deux, on évalue les produits $F(x_A) * F(x_0)$ et $F(x_0) * F(x_b)$.

La racine x_r se trouve donc séparée dans un des 2 intervalles, moitié de l'intervalle initial $[xa, xb]$. L'intervalle $[xa, x_0]$ ou $[x_0, xb]$ est maintenant désigné $[xa_1, xb_1]$.

On remarque que $|x_r - x_0| < \left|\frac{xa - xb}{2}\right|$

En poursuivant ainsi la dichotomie, nous obtenons une suite de valeurs approchées de la racine x_r.

$$x_0 = \frac{xa + xb}{2} \; ; \quad x_1 = \frac{xa_1 + xb_1}{2} \; ; \; \ldots ; \; x_n = \frac{xa_n + xb_n}{2}$$

telles que $\left| x_a - x_n \right| < \left|\frac{xa + xb}{2^{n+1}}\right|$

N.B. La méthode dichotomique converge si les valeurs initiales x_a et x_b encadrent au moins une des racines de la fonction F.

b.2. Fonction de l'interface

PROCEDURE GETCONCEQUIL

Arguments : - NR : le nombre de réactifs.
- NC : le nombre de composés.
- K : valeur de la constante d'équilibre.

- pour tout $1 \leq i \leq NC$: TCS [i] = le coefficient stoechiométrique du ième composé.
- pour tout $1 \leq j \leq NR$: CI [j] = la concentration initiale du jème réactif.

Précondition : pour tout $1 \leq j \leq NR$: CI [j] $\neq 0$ et TCS [j] = le coefficient stoechiométrique du jème réactif.
pour tout $NR+1 \leq i \leq NC$: TCS [i] = le coefficient stoechiométrique du ième produit.

Résultat : pour tout $1 \leq i \leq NC$: CE [i] = la concentration à l'équilibre du ième composé.

b.3. Algorithmes

PROCEDURE GETCONCEQUIL (TCS, CI, NR, NC, K, CE)

```
DEBUT
    VALINTERVAL (TCS, CI, NR, XA, XB);
    SI F(X_A) * F(X_B) < 0 ALORS
       DICHO (F, XA, XB, RAC) ;
       POUR I ← 1 JUSQU'A NR FAIRE
            CE [I] ← (CI [I] -(TCS [I] * RAC));
       POUR J ← NR+1 JUSQU'A NC FAIRE
            CE [J] ← TCS [J] * RAC
    SINON ERREUR;
FIN
```

PROCEDURE VALINTERVAL (TCS,CI,NR,XA,XB);

Arguments : - NR : le nombre de réactifs.
pour tout 1 <= i <= NR : CI[i] = la concentration initiale du i eme réactif.
et TCS[i] = le coefficient stoechiométrique du i eme réactif.

Précondition : pour tout 1 = < i < = NR : CI [i] ≠ 0

Résultat : XA,XB : les 2 bornes de l'intervalle où il faut rechercher la racine.

Postcondition : XA = 0.
XB = MINIMUM (CI [i] /TCS [i])
pour tout 1 < = i < = NR.

```
DEBUT
    POUR I ← 1 JUSQU'A NR FAIRE
        VAL [ I ] ← CI [ I ] /TCS [ I ] ;
    XA ← 0;
    XB ← MINIMUM (VAL,NR);
FIN
```

PROCEDURE DICHO (F,XA,XB,RAC)

Arguments : - F : la fonction dont on recherche la racine.
- XA,XB : les 2 bornes de l'intervalle où il faut rechercher la racine.

Précondition : F(XA) * F(XB) < 0.

Résultat : RAC : une des racines de la fonction F.

Postcondition : | XA-XB | <= PRECISION et XA <=RAC <=XB

```
DEBUT
    S ← F(XA) < 0 ;
    REPETER
        X ← (XA+XB)/2;
        Z ← F(X);
        SI (Z < 0) = S alors
                    A ← X
        sinon       B ← X
    JUSQU'A CE QUE ABS(XA-XB) ≤ PRECISION ;
    RAC ← X ;
FIN
```

II. MODULE DEMARCHE INDUCTIVE

A. Module "INFERER"

1. Procédure de l'interface

PROCEDURE PINFER

Arguments : - NV : le nombre de variables que comporte la relation recherchée.

- pour tout $1 \leq i \leq NV$: TVA [i] = un identificateur de variable.
- NP : le nombre de points nécessaires pour pouvoir tracer un graphique repésentatif de la variation relative de 2 variables.

Précondition : - pour tout $1 \leq i, j \leq NV$ et $i \neq j$:

TVA [i] ≠ TVA [j].

Résultat : - pour tout $1 \leq i \leq 2^{(nv-1)}-1$: TEQ [i] = une équation indispensable pour retrouver la loi existant entre les NV variables.

---

2. Conception

Nous avons déterminé lors du second chapitre que le nombre d'expériences NBEXP nécessaires pour retrouver inductivement une relation $f(v_1, v_2, \ldots, v_n) = 0$ existant entre n variables était égal à

$$NBEXP = np^{(n-1)}$$

On s'aperçoit dans le tableau ci-dessous que, lorsque n augmente, le nombre d'expériences qu'il aura lieu de réaliser devient rapidement très important.

np \ n	2	3	4	5
4	4	16	64	256
6	6	36	216	1296
8	8	64	512	4096
10	10	100	1000	10000
12	12	144	1728	20736

C'est pourquoi nous nous sommes volontairement limité à 4 variables au maximum.

Nous allons maintenant décrire successivement la méthode pour retrouver la relation $f(v_1, v_2, \ldots, v_n) = 0$ avec n = 2, 3 et 4.

---

2 VARIABLES

$C = \{v_1, v_2\}$; NVC = 2.(*NVC = le nombre de variables dans C*)

a; Choix de la variable en ordonnée : $vy \in C$.

$C = C - \{vy\}$; NVC = 1.

b; La variable en abscisse $vx = v \in C$.

c; Pour tout v_i appartenant à C : fixer le domaine de variation.

$C = C - \{vx\}$; NVC = 0.

d; Etablir une relation linéaire entre vx et vy.

$$vy = p1.vx + O1$$

avec p1, O1 = constantes.

- réalisation d'un graphique vy/vx.

—

3 VARIABLES

$C = \{v_1, v_2, v_3\}$; NVC = 3.

a; Choix de la variable en ordonnée : $vy \in C$.

$C = C - \{vy\}$; NVC = 2.

b; Choix de la variable en abscisse : $vx \in C$.

c; Pour tout v_i appartenant à C : fixer le domaine de variation.

$C = C - \{vx\}$; NVC = 1.

d; Etablir une relation linéaire entre vx et vy.

$$vy = p1.vx + O1$$

avec p1, O1 = f(VAR3)

où VAR3 = la variable restant dans C.

- réalisation de NP graphiques vy/vx avec VAR3 = Cste.
- mémorisation de NP valeurs de p1 et NP valeurs de O1.

pour tout $1 \leq i \leq NP$: LA1 $[i]$: valeur de p1

MU1 $[i]$: valeur de O1

pour la ième valeur de VAR3.

e; Etude de la variation de p1/VAR3 et O1/VAR3.

$C = C - \{VAR3\}$; NVC = 0.

e.1; Etablir une relation linéaire entre p1 et VAR3.

$$p1 = p2.VAR3 + O2$$

avec p2, O2 = constantes.

- utilisation des NP valeurs de p1 mémorisées dans LA1.
- réalisation d'un graphique p1/VAR3.

e.2; Etablir une relation linéaire entre O1 et VAR3.

$$O1 = p3.VAR3 + O3$$

avec p3, O3 = constantes.

- utilisation des NP valeurs de O1 mémorisées dans MU1.
- réalisation d'un graphique O1/VAR3.

N.B. La ième valeur de la variable de numéro v = $(MIN[v] + \frac{(i-1)*CMAX[v] - CMIN[v]}{(NP-1)}$

où CMIN $[v]$ = valeur minimale

CMAX $[v]$ = valeur maximale

que prendra la variable de numéro v.

—

4 VARIABLES

$C = \{ v_1, v_2, v_3, v_4 \}$; NVC = 4.

a; Choix de la variable en ordonnée : $vy \in C$.

$C = C - \{ vy \}$; NVC = 3.

b; Choix de la variable en abscisse : $vx \in C$.

c; Pour tout v_i appartenant à C : fixer le domaine de variation.

$C = C - \{ vx \}$; NVC = 2.

d; Etablir une relation linéaire entre vx et vy.

vy = pl.vx + Ol

avec pl, Ol = f(VAR4, VAR5)

où VAR4 : la première variable de C.

où VAR5 : la seconde variable de C.

- réalisation de NP^2 graphiques vy/vx avec VAR4, VAR5 =Cstes.
- mémorisation de NP^2 valeurs de pl et NP^2 valeurs de Ol.

pour tout $1 \leq i, j \leq NP$: LA2 [i, j] : valeur de pl,

MU2 [i, j] : valeur de Ol

pour la ième valeur de VAR4 et

pour la jème valeur de VAR5.

e; Choix entre VAR4 et VAR5 : $VXC \in C$.

si VXC = VAR4 alors VAR3 = VAR5; $C = C - \{ VAR4 \}$.

si VXC = VAR5 alors VAR3 = VAR4; $C = C - \{ VAR5 \}$.

NVC = 1.

f; Etude de la variation de p1/VXC et O1/VXC.

f.a; Etablir une relation linéaire entre p1 et VXC.

$$p1 = p2.VXC + O2$$

avec p2, O2 = f(VAR3).

- si VXC = VAR4 alors utilisation des valeurs de p1 mémorisées dans les NP lignes de LA2.
- si VXC = VAR5 alors utilisation des valeurs de p1 mémorisées dans les NP colonnes de LA2.
- réalisation de NP graphiques p1/VXC avec VAR3 = Cste.
- mémorisation des NP valeurs de p2 et NP valeurs de O2.

Pour tout $1 \leq i \leq NP$: LA1 $[i]$: valeur de p2,
MU1 $[i]$: valeur de O2
pour la ième valeur de VAR3.

f.a.1; Etude de la variation de p2/VAR3 et O2/VAR3.

C = C- {VAR3} ; NVC = 0.

f.a.1.1; Etablir une relation linéaire entre p2 et VAR3.

$$p2 = p3.VAR3 + O3$$

avec p3, O3 = constantes.

- utilisation des NP valeurs de p2 mémorisées dans le tableau LA1.
- réalisation d'un graphique p2/VAR3.

f.a.1.2; Etablir une relation linéaire entre O2 et VAR3.

$$O2 = p4.VAR3 + O4$$

avec p4, O4 = constantes.

- utilisation des NP valeurs de O2 mémorisées dans le tableau MU1.
- réalisation d'un graphique O2/VAR3.

C = C + $\{$VAR3$\}$; NVC = 1.

f.b; Etablir une relation linéaire entre p1 et VXC.

$$O1 = p5.VXC + O5$$

avec p5, O5 = f(VAR3).

- si VXC = VAR4 alors utilisation des valeurs de O1 mémorisées dans les NP <u>lignes</u> de MU2.
- si VXC = VAR5 alors utilisation des valeurs de O1 mémorisées dans les NP <u>colonnes</u> de MU2.
- réalisation de NP graphiques O1/VXC avec VAR3 = Cste.
- mémorisation des NP valeurs de p5 et des NP valeurs de O5.

pour tout $1 \leq i \leq NP$: LA1 $[i]$: valeur de p5,
MU1 $[i]$: valeur de O5
pour la ième valeur de VAR3.

f.b.1; Etude de la variation de p5/VAR3 et O5/VAR3.

C = C- $\{$VAR3$\}$; NVC = O.

f.b.1.1; Etablir une relation linéaire entre p5 et VAR3.

$$p5 = p6.VAR3 + O6$$

avec p6, O6 = constantes.

- utilisation des NP valeurs de p5 mémorisées dans le tableau LA1.
- réalisation d'un graphique p5/VAR3.

f.b.1.2; Etablir une relation linéaire entre O5 et VAR3.

$$O5 = p7.VAR3 + O7$$

avec p7, O7 = constantes.

- utilisation des NP valeurs de O5 mémorisées dans le tableau MU1.
- réalisation d'un graphique O5/VAR3.

---

3. Algorithmes

* Les différentes variables intervenant dans la relation $f(v_1, v_2, \ldots, v_n) = 0$ sont mémorisées dans la structure de données

TABVAR = array [1..NBMAXVAR] of VARIABLE.

tel que TVA [i] = identificateur de la ième variable avec TVA du type TABVAR.

Nous avons défini sur cette structure 3 primitives:

1° FONCTION EXVATA

Arguments : V = un identificateur.

pour tout $1 \leq i \leq NV$:

TVA [i] = un identificateur.

Résultat : - VRAI s'il existe $1 \leq i \leq NV$: TVA [i] = V

alors NUM = i.

- FAUX sinon.

2° FONCTION VARIA

Arguments : - NUM = numéro d'identificateur.

- pour tout $1 \leq i \leq NV$:

TVA [i] = un identificateur.

Précondition : $1 \leq NUM \leq NV$.

Résultat : identificateur de numéro NUM.

3° FONCTION NUMERO

Arguments V = identificateur.

pour tout $1 \leq i \leq NV$

TVA [i] = un identificateur.

Précondition : EXVATA (V, TVA, NV, NUM).

Résultat : numéro de l'identificateur V (=NUM).

---

* Afin de pouvoir :

- choisir la variable en ordonnée et en abscisse,
- fixer les domaines de variation,
- savoir quelles sont les différentes variables maintenues constantes lors d'une expérience,

nous avons défini la structure de données suivante :

CRIBLE = array [1..NBMAXVAR] of integer

tel que NO [i] = { 1 si VARIA (i, TVA) appartient au crible.
                 { 0 sinon.

avec NO du type CRIBLE.

Nous avons défini 4 primitives sur cette structure de données :

1° FONCTION EXVACR

Arguments : - NUM : un numéro d'identificateur.
- pour tout 1 <= i <= NV : NO [i] = 0 ou 1.

Précondition : 1 <= NUM <= NV.

Résultat : - VRAI si NO [NUM] = 1.
- FAUX sinon.

2° FONCTION RETIRER

Arguments : - NUM : un numéro d'identificateur.
- pour tout 1 <= i <= NV : NO [i] = 0 ou 1.

Précondition : EXVACR (NUM, NO) = vrai
1 <= NUM <= NV.

Postcondition : EXVACR (NUM, NO) = faux

3° FONCTION REMETTRE

Arguments : - NUM : un numéro d'identificateur.
- pour tout 1 <= i <= NV : NO [i] = 0 ou 1.

Précondition : EXVACR (NUM, NO) = faux

$1 \leq NUM \leq NV$.

Postcondition : EXVACR (NUM, NO) = vrai

4° PROCEDURE INITCRIBLE

Arguments : - NV : le nombre de variables.

- pour tout $1 \leq i \leq NV$: NO [i] = 0 ou 1.

Résultat : - pour tout $1 \leq i \leq NV$: NO [i] = 1.

Enfin, les différents points (X [i] , Y [i])nécessaires pour tracer le graphique sont obtenus de la manière suivante :

$$X[i] = CMIN[i] + \frac{((i-1)}{(NP-1)} * (CMAX[i] - CMIN[i]);$$

X [i] = VAL [NUMERO (VX, TVA, NV)] ;

et Y [i] = VAL [NUMERO (VY, TVA, NV)];

Y [i] = f (VAL [NUMERO (Vj, TVA, NV)]) pour tout $v_j \neq VY$

où VAL [j] = la valeur attribuée à la variable de numéro j.

N.B. La fonction permettant la détermination de Y [i] en fonction de VAL [j] tel que $i \leqslant j \leqslant NV$ et $i \neq j$ ainsi que la procédure permettant de fixer le domaine de variation d'une variable seront localisées dans un module distinct : "FONCTION A INFERER".

```
PROCEDURE PINFER (NV, NP, TVA, TEQ)

DEBUT
    REUSSI ← FALSE;
    REPETER
        INITCRIBLE (NO, NV);
        SUIVANT (NV) FAIRE
            1 : ECRIRE ('ERREUR');
            2 : VAR 2;
            3 : VAR 3;
            4 : VAR 4;
            >4 : ECRIRE ('NON ENVISAGE')
     JUSQU'A REUSSI;
FIN
```

```
PROCEDURE VAR 2
DEBUT     (*NOMBRE DE VARIABLES DANS NO : NVNO = 2*)
    CHOIX ('Y', TVA, NV, NO, VY, NY);
    RETIRER (NY, NO);     (*NVNO = 1*)
    NX ← GETVAR (NO, NV); VX ← VARIA (NX, TVA);
    DOMVAR (TVA, NO, NV, NP, CMIN, CSTEP);
    RETIRER (NX, NO);      (*NVNO = 0*)
    NUMEQ ← 0;
    SI GETVAR2    (VX, VY, CMIN, CSTEP, NP, CX, CY) ALORS
                SI TEST2  (CX, CY, NP, VX, VY.) ALORS
                        REUSSI ← TRUE;
FIN
```

PROCEDURE VAR 3

```
DEBUT    (*NVNO = 3*)
    CHOIX ('Y', TVA, NV, NO, VY, NY);
    RETIRER (NY, NO);      (*NVNO = 2*)
    CHOIX ('X', TVA, NV, NO, VX, NX);
    DOMVAR (TVA, NV, NO, NP, CMIN, CSTEP);
    RETIRER (NX, NO);      (*NVNO = 1*)
    NUMEQ ← 0;
    SI GETVAR3  (VX, VY, CMIN, CSTEP, NP, LA1, MU1) ALORS
            NUMEQ ← NUMEQ+1;
            PE ← CONCAT ('P', CAR(NUMEQ));
            OR ← CONCAT ('O', CAR(NUMEQ));
            EQUATION (VX, VY, PE, OR, TEQ [NUMEQ] );
            ECRIRE (TEQ [NUMEQ] );
            SI TEST3  (PE, OR, LA1, MU1, NP) ALORS
                        REUSSI ← TRUE
FIN
```

PROCEDURE VAR 4

```
DEBUT    (*NVNO = 4*)
    CHOIX ('Y', TVA, NV, NO, VY, NY);
    RETIRER (NY, NO);      (*NVNO = 3*)
    CHOIX ('X', TVA, NV, NO, VX, NX);
    DOMVAR (TVA, NV, NO, NP, CMIN, CSTEP);
    RETIRER (NX, NO);      (*NVNO = 2*)
    NUMEQ ← 0;
```

```
   SI GETVAR4  (VX, VY, CMIN, CSTEP, NP, LA2, MU2) ALORS
            NUMEQ ← NUMEQ + 1;
            PE ← CONCAT ('P', CAR (NUMEQ));
            OR ← CONCAT ('O', CAR (NUMEQ));
            EQUATION (VX, VY, PE, OR, TEQ [NUMEQ] );
            ECRIRE (TEQ [NUMEQ] );
            SI TEST4  (PE, OR, LA2, MU2, NP) ALORS
                        REUSSI ← TRUE.
FIN
```

FONCTION VARIA (NUM, TVA)

```
DEBUT
    VARIA ← TVA [NUM]
FIN
```

FONCTION NUMERO (V, TVA, NV)

```
DEBUT
    POUR I←1 JUSQU'A NV FAIRE
            SI TVA [I] = V ALORS NUMERO← I;
FIN
```

FONCTION EXVATA (V, TVA, NV, NUM)

```
DEBUT
    EXVATA ← FALSE ;
    I ← 0 ;
    TROUVE ← FALSE ;
```

```
    TANT QUE I < NV ET PAS TROUVE FAIRE
            I ← I + 1;
            SI V = TVA [I] ALORS
                TROUVE ← TRUE;
                EXVATA ← TRUE;
                NUM ← I
FIN
```

PROCEDURE INITCRIBLE (NO, NV)

```
DEBUT
    POUR I ← 1 JUSQU'A NV FAIRE
            NO [I] ← 1;
FIN
```

FONCTION EXVACR (NO, NUM)

```
DEBUT
    EXVACR ← FALSE;
    SI NO [NUM] = 1 ALORS
            EXVACR ← TRUE;
FIN
```

PROCEDURE RETIRER (NUM, NO)

```
DEBUT
    NO [NUM] ← 0
FIN
```

PROCEDURE REMETTRE (NUM, NO)

```
DEBUT
    NO [NUM] ← 1
FIN
```

FONCTION GETVAR (NO, NV)

```
DEBUT
    I ← 0;
    TROUVE ← FALSE;
    TANT QUE I < NV ET PAS TROUVE FAIRE
        I ← I + 1;
        SI EXVACR (NO, I) ALORS
                GETVAR ← I;
                TROUVE ← TRUE;
FIN
```

PROCEDURE CHOIX (C, TVA, NV, NO, V, NUM)

Arguments : - C : 'X' ou 'Y'.

- pour tout 1 <=i <=NV : TVA [i] = un identificateur.
- pour tout 1 <=i <=NV : NO [i] = 0 ou 1.

Précondition : il existe au moins un 1 <=i <=NV :
EXVACR (i, NO) = vrai.

Résultat : - V : un identificateur de variable.
- NUM : NUMERO (V, TVA, NV)

Postcondition : EXVACR (NUM, NO) et
EXVATA (V, TVA, NV) = vrai.

```
DEBUT
    FIN ← FALSE;
    REPETER
          ECRIRE (C);
          LIRE (V);
          SI EXVATA (V, TVA, NV, NUM) ALORS
             SI EXVACR (NO, NUM) ALORS
                    FIN ← TRUE
    JUSQU'A FIN;
FIN
```

PROCEDURE DOMVAR (TID, NO, NID, NP, CMIN, CSTEP)

Arguments : - NP : le nombre de points.
- pour tout 1 <=i <=NV : TVA [i] = un identificateur de variable.
- pour tout 1 <=j <=NV : NO [i] = 0 ou 1.

Résultat : pour tout i tel que EXVACR (i,NO) = vrai.

alors

- CMIN [i] : la valeur initiale pour la variable de numéro i.
- CSTEP [i] : le pas à appliquer.

Postcondition : pour tout i tel que EXVACR (i, NO) = vrai

alors

- CSTEP [i] > 0.

```
DEBUT
    POUR I ← 1 JUSQU'A NID FAIRE
        SI EXVACR (NO, I) ALORS
                DVVH (TID [I], CMIN [I], CMAX)
            OU DVGW (TID [I], CMIN [I], CMAX);
                CSTEP [I] ← (CMAX - CMIN [I])/(NP-1);
FIN
```

FONCTION GETVAR2 (VX, VY, CMIN, CSTEP, NP, CX, CY)

Arguments : - VX : l'identificateur de la variable en abscisse.

- VY : l'identificateur de la variable en ordonnée.
- pour tout 1 <=i <=NV et i ≠ VY :
  - CMIN [i] : la plus petite valeur pour la variable VARIA(i, TVA).
  - CSTEP [i] : le pas à appliquer lors de la variation de VARIA (i, TVA).

Précondition : pour tout 1 <=NUM <=NV : EXVACR (NUM, NO) = faux.

Résultat : - vrai alors

pour tout $1 \leq i \leq NP$: $CX[i] = VAL[i] =$

$CMIN[NUMERO(VX, TVA, NV)] + ((i-1)*$

$CSTEP[NUMERO(VX, TVA, NV)]$).

$CY[i] = VALY(NUMERO$

$(VY, TVA, NV)], VAL)$.

```
DEBUT    (*NVNO = 0*)
    GETVAR2 ← TRUE;
    CALCUL (VAL, NX, NY, CMIN, CSTEP, CX, CY, NP);
FIN
```

PROCEDURE CALCUL (VAL, NX, NY, CMIN, CSTEP, X, Y, N)

Arguments : - NX : le numéro de la variable en abscisse.

- NY : le numéro de la variable en ordonnée.
- pour tout $1 \leq i \leq NV$ et $i \neq NX \neq NY$:

  $VAL[i]$: valeur de la variable de numéro i.
- pour tout $1 \leq i \leq NV$ et $i \neq VY$:

  $CMIN[i]$: la plus petite valeur pour la variable VARIA(i, TVA)

  $CSTEP[i]$: le pas à appliquer lors de la variation de VARIA(i,TVA)

Résultat : - pour tout $1 \leq i \leq N$: $X[i] = VAL[NX] =$

$CMIN[NX] + ((i-1)*CSTEP[NX])$.

$Y[i] = VALY(NY)$.

```
DEBUT
    POUR I ← 1 JUSQU'A N FAIRE
          VAL [NX] ← CMIN [NX] + ((I-1)*CSTEP [NX] );
          X [I] ← VAL [NX] ;
          Y [I] ← VALY (VAL, NY);
FIN
```

FONCTION VALY (VAL, NY)

Arguments : - NY = numéro de variable en ordonnée.

- pour tout $1 \leq i \leq NV$ et $i \neq NY$:

- VAL [i] : valeur de la variable de numéro i.

Précondition : $1 \leq NY \leq NV$.

Résultat : - VALY = valeur de la variable de numéro NY.

Règle : - VALY est déterminé grâce à la relation $f(v_1, v_2, \ldots, v_n) = 0$ reprise dans le module "fonction à inférer"

```
DEBUT
    VALY ← GULWAAGE (NC, NR, NY, K, TCS, VAL)
 OU VALY ← VANHOOFT (TID, NID, VD, DHO, DSO, VAL)
FIN
```

FONCTION TEST2 (X, Y, N, VY, VX)

Précondition : pour tout 1 <=NUM <=NV : EXVACR (NUM, NO) = faux.

Arguments : - Pour tout 1 <=i <=N : X [i] = un réel représentant une valeur de VX.

Y [i] = un réel représentant une valeur de VY pour une valeur X [i] de VX.

- VX : l'identificateur de la variable en abscisse.
- VY : l'identificateur de la variable en ordonnée.

Résultat : - vrai si on a trouvé une relation linéaire entre VX et VY.
- faux sinon.

```
DEBUT    (*NVNO = 0*)
    TEST2 ← FALSE;
    SI TROUVERDROITE (X, Y, N, VX, VY, PENTE, ORIGINE,
    CHX, CHY) ALORS
                    NUMEQ ← NUMEQ + 1;
                    EQUATION (VX, VY, CAR(PENTE), CAR(ORIGINE);
                                TEQ [NUMEQ] );
                    ECRIRE (TEQ [NUMEQ] );
                    TEST2 ← TRUE;
FIN
```

FONCTION TROUVERDROITE (X, Y, N, VX, VY, P, CO, CHX, CHY)

Arguments : - VX : l'identificateur de la variable en abscisse.

- VY : l'identificateur de la variable en ordonnée.
- Pour tout i : $1 \leq i \leq N$: X [i] = un réel représentant une valeur de VX.

Y [i] = un réel représentant une valeur de VY pour une valeur X [i] de VX.

Résultats : - vrai si on a trouvé une relation linéaire entre VX et VY.

- P : la pente de la droite.
- CO : l'ordonnée à l'origine de la droite.
- VX : concat (f(CHX), VX).
- VY : concat (f(CHY), VY).

où f(c) = l'identificateur de la fonction de transformation de numéro c.

- CHX : le numéro de la fonction de transformation de l'échelle des X pour lequel on a trouvé une relation linéaire.
- CHY : le numéro de la fonction de transformation de l'échelle Y pour lequel on a trouvé une relation linéaire.

- faux sinon.

```
DEBUT
    TROUVERDROITE ← FALSE;
    FINI ← FALSE;
    XC ← X; YC ← Y;
    VXC ← VX; VYC ← VY;
    CHX ← 1; CHY ← 1;          (* TRANSFORMATION IDENTITE*)
```

```
REPETER
        SI EXISTDROITE  (XC, YC, N, VXC, VYC, P, CO)
        ALORS
                        FINI ← TRUE;
                        TROUVERDROITE ← TRUE
        SINON TRANSECHEL (X, Y, N, VX, VY, XC, YC, VXC,
                                VYC, CHX, CHY)
  JUSQU'A FINI;
  VX ← VXC;
  VY ← VYC;
FIN
```

FONCTION EXISTDROITE (X, Y, N, VX, VY, P, CO)

Arguments : - VX : l'identificateur de la variable en abscisse.

- VY : l'identificateur de la variable en ordonnée.
- Pour tout $1 \leq i \leq N$: $X[i]$ = un réel représentant une valeur de VX.

  $Y[i]$ = un réel représentant une valeur de VY pour une valeur $X[i]$ de VX.

Résultat : - vrai s'il existe une relation linéaire entre VX et VY alors
  - P: la pente de la droite.
  - CO : l'ordonnée à l'origine de la droite.
- faux sinon.

```
DEBUT
    EXISTDROITE ← FALSE;
    DESSINGRAF (X, Y, N, VX, VY, PC1, PC2);
    REGLIN (X, Y, N, AR, BR, CR, CC);
    PENTORIG (AR, BR, CR, P, CO);
    DROITEREG  (AR, BR, CR);
    SI CC = 1 ALORS EXISTDROITE ← TRUE
FIN
```

FONCTION GETVAR3 (VX, VY, CMIN, CSTEP, N , LA1, MU1)

Arguments : - VX : l'identificateur de la variable en abscisse.

- VY : l'identificateur de la variable en ordonnée.
- pour tout 1 ≤ i ≤ NV et i ≠ VY :
  - CMIN [i] : la plus petite valeur pour la variable VARIA(i,TVA).
  - CSTEP [i] : le pas à appliquer lors de la variation de VARIA (i, TVA).

Précondition : il existe un seul 1 ≤ NUM3 ≤ NV :
EXVACR(NUM3, NO) = vrai .

Résultat : - vrai si on a trouvé une relation linéaire entre VX et VY alors

pour tout 1 ≤ j ≤ N : LA1 [j] = valeur de la pente pl,

MU1 [j] = valeur de
l'ordonnée à l'origine o1 de l'équation
VY = p1 * VX + o1 pour la jème valeur de
VARIA (NUM3, TVA)
(VAL [NUM3] = CMIN [NUM3] + ((j-i)*CSTEP [NUM3]))

- faux sinon.

```
DEBUT   (*NVNO = 1*)
    J ← 1;
    NUM3 ← GETVAR (NO, NV);
    VAL [NUM3] ← CMIN [NUM3] ;
    CALCUL (VAL, NX, NY, CMIN, CSTEP, CX, CY, N );
    SI TROUVERDROITE (CX, CY, N , VX, VY, LA1 [ J] , MU1 [ J],
                      CHX, CHY) ALORS
        CONTINUE ← TRUE;
        TANT QUE CONTINUE ET J < N FAIRE
            J ← J + 1;
            VAL [NUM3] ← CMIN [NUM3] +((J-1) *
            CSTEP [NUM3] );
            CALCUL (VAL, NX, NY, CMIN, CSTEP, CX,
                    CY, N );
            SI REALI (CHX, CX, N) ET REALI (CHY,
            CY, N) ALORS
                TRANSFO (CHX, CX, N, CX);
                TRANSFO (CHX, CX, N, CX);
                SI PAS EXISTDROITE (CX, CY, N,
                VX, VY, LA1 [ J] , MU1 [ J] ) ALORS
                    CONTINUE ← FALSE
```

```
                    SINON CONTINUE ← FALSE
    SINON CONTINUE ← FALSE;
    GETVAR3 ← CONTINUE;
FIN
```

FONCTION TEST3 (P, O, L1, M1, N)

Précondition : il existe un seul 1 <=NUM3 <=NV :
EXVACR (NUM3, NO) = vrai.

Arguments : - P : identificateur pour la pente.
- O : identificateur pour l'ordonnée à l'origine.
- pour tout 1 <=j <=N : L1 [j] = valeur de la pente ,
M1 [j] = valeur de l'ordonnée à l'origine de l'équation VY = P * VX + O pour la jème valeur de VARIA (NUM3, TVA).
(VAL [NUM3] = CMIN [NUM3] + ((j-1) * CSTEP [NUM3])).

Résultat : - vrai si on a trouvé une relation linéaire entre
- P et VARIA (NUM3, TVA).
- O et VARIA (NUM3, TVA).
- faux sinon.

```
DEBUT   (*NVNO = 1*)
    TEST3 ← FALSE;
    NXC ← GETVAR (NO, NV);
    VXC ← VARIA (NXC, TVA);
    VXCM ← VXC;
    RETIRER (NXC, NO);    (*NVNO = 0*)
    SI GETMEM2 (L1, N, P, VXC, CX, CY) ALORS
       SI TEST2 (CX, CY, N, P, VXC) ALORS
          SI GETMEM2 (M1, N,0 , VXCM, CX, CY) ALORS
             SI TEST2 (CX, CY, N, 0, VXCM) ALORS
                          TEST3 ← TRUE;
    REMETTRE (NXC, NO)    (*NVNO = 1*)
FIN
```

FONCTION GETMEM2 (T1, N, VY, VX, CX, CY)

Précondition : pour tout $1 \leq NUM \leq NV$: EXVACR(NUM,NO) = faux.

Arguments : - VX : l'identificateur de la variable en abscisse.

- VY : l'identificateur de la variable en ordonnée.
- pour tout $1 \leq j \leq N$: T1 [j] = valeur de VY pour la jème valeur de VARIA(NUM3,TVA). (VAL [NUM3] = CMIN [NUM3] + ((j-1)* CSTEP [NUM3]))

Résultat : - vrai alors

pour tout $1 \leq i \leq N$: CX [i]= CMIN [NUMERO (VX, TVA, NV)] + ((i-1) * CSTEP [NUMERO (VX, TVA, NV)])

CY [i] = T1 [i].

- faux sinon.

```
DEBUT   (*NVNO = 0*)
    GETMEM2 ← TRUE;
    NX ← NUMERO (VX, TVA, NV);
    POUR I ← 1 JUSQU'A N FAIRE
            CX [I] ← CMIN [NX] + ((I-1) * CSTEP [NX] );
            CY [I] ← T1 [I] ;
FIN
```

FONCTION GETVAR4 (VX, VY, CMIN, CSTEP, N , LA2, MU2)

Arguments : - VX : l'identificateur de la variable en abscisse.

- VY : l'identificateur de la variable en ordonnée.
- pour tout $1 \leq i \leq NV$ et $i \neq VY$:
  - CMIN [i] : la plus petite valeur pour la variable VARIA(i,TVA).
  - CSTEP [i] : le pas à appliquer lors de la variation de VARIA (i, TVA).

Précondition : il existe un seul $1 \leq NUM4 \leq NV$:
EXVACR (NUM, NO) = vrai.
et il existe un seul $1 \leq NUM5 \leq NV$:
EXVACR (NUM, NO) = vrai.
et NUM4 $\neq$ NUM5.

Résultat : - vrai si on a trouvé une relation linéaire entre VX et VY.

alors

pour tout $1 \leq j, k \leq N$:

- LA 2 [j, k] : valeur de la pente pl,
- MU 2 [j, k] : valeur de l'ordonnée à l'origine Ol de la droite d'équation VY = pl * VX + ol pour la jème valeur de VARIA (NUM4, TVA). (VAL [NUM4] = CMIN [NUM4] + (j-1) * CSTEP [NUM4])) et la kème valeur de VARIA (NUM5, TVA). (VAL [NUM5] = CMIN [NUM5] + ((k-1) * CSTEP [NUM5]))

- faux sinon.

```
DEBUT  (*NVNO = 2*)
    J ← 1;
    NUM 4 ← GETVAR (NO, NV);
    RETIRER (NUM4, NO);   (*NVNO = 1*)
    VAL [NUM4] ← CMIN [NUM4] ;
    K ← 1;
    NUM5 ← GETVAR (NO, NV);
    VAL [NUM5] ← CMIN [NUM5] ;
    REMETTRE (NUM4, NO);   (*NVNO = 2*)
    CALCUL (VAL, NX, NY, CMIN, CSTEP, CX, CY, N );
    SI TROUVERDROITE (CX, CY, N, VX, VY, LA2 [J , K ] , MU2
                      [J , K ], CHX, CHY) ALORS
          CONTINUE ← TRUE;
          TANT QUE CONTINUE ET J ≤ N FAIRE
```

```
                    TANT QUE CONTINUE ET K < N FAIRE
                        K ← K + 1;
                        VAL [NUM5] ← CMIN [NUM5] + ((K-1) *
                        CSTEP [NUM5] );
                        CALCUL (VAL, NX, NY, CMIN, CSTEP,
                        CX, CY, N );
                        SI REALI (CHX, CX, N) ET REALI (CHY,
                        CY, N) ALORS
                            TRANSFO (CHX, CX, N, CX);
                            TRANSFO (CHY, CY, N, CY);
                            SI PAS EXISTDROITE (CX, CY, N, VX,
                            VY, LA2 [J, K], MU2 [J, K] ) ALORS
                                    CONTINUE ← FALSE;
                        SINON CONTINUE ← FALSE;
                    K ← 0;
                    J ← J + 1;
                    VAL [NUM4] ← CMIN [NUM4] + ((J-1) *
                    CSTEP [NUM4] )
    SINON CONTINUE ← FALSE;
    GETVAR4 ← CONTINUE;
FIN
```

FONCTION TEST4 (P, O, L2, M2, N)

Précondition : - il existe un seul 1 < =NUM4 < =NV :

EXVACR (NUM, NO) = vrai.

et il existe un seul 1 < =NUM5 < =NV :

EXVACR (NUM5, NO) = vrai.

et NUM4 ≠ NUM5.

Arguments : - P : identificateur pour la pente.

- O : identificateur pour l'ordonnée à l'origine.
- pour tout 1 ≤ j, k ≤ NP :

  - L2 [j, k] : valeur de la pente,
  - M2 [j, k] : valeur de l'ordonnée à l'origine de la droite d'équation VY = p1 * VX + o1 pour la jème valeur de VARIA (NUM4, TVA). (VAL [NUM4] = CMIN [NUM4] + ((j-1) * CSTEP [NUM4])) et la kème valeur de VARIA (NUM5, TVA). (VAL [NUM5] = CMIN [NUM5] + ((k-1) * CSTEP [NUM5]))

Résultat : - vrai si on a trouvé une relation linéaire entre

  - P et VARIA (NUM4, TVA).
  - O et VARIA (NUM4, TVA).
  - P et VARIA (NUM5, TVA).
  - O et VARIA (NUM5, TVA).

- faux sinon.

```
DEBUT   (*NVNO = 2*)
    TEST4 ← FALSE;
    CHOIX ('X', TVA, NV, NO, VXC, NXC);
    VXCM ← VXC;
    RETIRER (NXC, NO);    (*NVNO = 1*)
```

```
   SI NXC = NUM4 ALORS NUM3 ← NUM5
   SINON NUM3 ← NUM4;
   SI GETMEM3 (L2, N, P, VXC, L1, M1) ALORS
           NUMEQ ← NUMEQ + 1;
           PE ← CONCAT ('P', CAR(NUMEQ));
           OR ← CONCAT ('O', CAR(NUMEQ));
           EQUATION (VXC, P, PE, OR, TEQ [NUMEQ] );
           ECRIRE (TEQ [NUMEQ] );
           SI TEST3 (PE, OR, L1, M1, N) ALORS
              SI GETMEM3 (M2,N, O, VXCM, L1, M1) ALORS
                   NUMEQ ← NUMEQ + 1;
                   PE ← CONCAT ('P', CAR(NUMEQ));
                   OR ← CONCAT ('O', CAR(NUMEQ));
                   EQUATION (VXCM, O, PE, OR, TEQ [NUMEQ] );
                   ECRIRE (TEQU [NUMEQ] );
                   SI TEST3 (PE, OR, L1, M1, N) ALORS
                            TEST4 ← TRUE;
   REMETTRE (NXC, NO)   (*NVNO = 2*)
FIN
```

FONCTION GETMEM3 (T2,N, VX, VY, L1, M1)

Précondition : il existe un seul 1 <=NUM3 <=NV :
EXVACR (NUM3, NO) = vrai.

Arguments : - VX : l'identificateur de la variable en abscisse.
- VY : l'identificateur de la variable en ordonnée.

- pour tout 1 ≤ =j, k ≤ =N :

  T2 [j, k] valeur de VY pour la jème valeur de VARIA (NUM, TVA)
  (VAL [NUM4] = CMIN [NUM4] + ((j-1) * CSTEP [NUM4])) et la kème valeur de VARIA (NUM4, TVA).
  (VAL [NUM5] = CMIN [NUM5] + ((k-1) * CSTEP [NUM5]))

Résultat : - vrai si on a trouvé une relation linéaire entre VX et VY.
alors

- pour tout 1 ≤ =i ≤ =N : L1 [i] = valeur de la pente,
  M1 [i] = valeur de l'ordonnée à l'origine de l'équation VY = P * VX + O pour la ième valeur de VARIA (NUM3, TVA).
  (VAL [NUM3] = CMIN [NUM3] + ((i-1) * CSTEP [NUM3]))

- faux sinon.

```
DEBUT   (*NVNO = 1*)
    I ← 1;
    NX ← NUMERO (VX, TVA, NV);
    POUR J ← I JUSQU'A N FAIRE
```

```
                CX [J] ← CMIN [NX] + ((J-1) * CSTEP [NX] );
                SI NX = NUM4 ALORS
                        CY [J] ← T2 [J, I]
                SINON CY [J] ← T2 [I, J] ;
        VAL [NUM3] ← CMIN [NUM3] ;
        SI TROUVERDROITE (CX, CY, N, VX, VY, L1 [I ], M1 [I] ,
                                CHX, CHY) ALORS
                CONTINUE ← TRUE;
                TANT QUE CONTINUE ET I < N FAIRE
                        I ← I + 1;
                        VAL [NUM3] ← CMIN [NUM3] + ((I-1) *
                        CSTEP [NUM3] );
                        J ← 0;
                        TANT QUE CONTINUE ET J < N FAIRE
                                J ← J + 1;
                                CX [J] ← CMIN [NX] + ((J-1) *
                                CSTEP [NX] );
                                SI NX = NUM4 ALORS
                                        CY [J] ← T2 [J, I]
                                SINON CY [J] ← T2 [I, J]
                                SI REALI (CHX, CX, N) ET REALI (CHY,
                                CY, N) ALORS
                                   TRANSFO (CHX, CX, N, CX);
                                   TRANSFO (CHY, CY, N, CY);
                                   SI PAS EXISTDROITE (CX, CY, N,VX
                                   VY, L1 [I] , M1 [I] ) ALORS
                                                CONTINUE ← FALSE;
                                SINON CONTINUE ← FALSE
        SINON CONTINUE ← FALSE
        GETMEM3 ← CONTINUE
FIN
```

B. Module "FONCTIONS-A-INFERER"

1. Conception

Ce module reprendra, pour chaque relation $f(v_1, v_2, \ldots v_n) = 0$ susceptible d'être retrouvée inductivement grâce au module "INFERER" ,

* une fonction qui déterminera la valeur de la variable v_i en fonction de la valeur attribuée aux variables v_j tel que $1 \leqslant j \leqslant n$ et $j \neq i$.

* une procédure qui permettra de fixer le domaine de variation de chacune des variables intervenant dans la relation $f(v_1, v_2, \ldots v_n) = 0$.

2. Fonctions et procédures de l'interface

a. Fonction GULWAAGE

Arguments : - K : valeur de la constante d'équilibre de la réaction.

- NR : le nombre de réactifs que comporte la réaction.
- NC : le nombre de composés que comporte la réaction .

pour tout $1 \leqslant i \leqslant NC$:

- TCO [i] : l'identificateur du ième composé.
- TCS [i] : le coefficient stoechiométrique du composé i.

Précondition : pour tout $1 \leqslant i \leqslant NR$: TCO [i] : un réactif.

$NR + 1 \leqslant i \leqslant NC$: TCO [i] : un produit.

$NC > NR$.

- ND : numéro du composé chimique pour lequel on doit déterminer la valeur de la concentration à l'équilibre.

Précondition : $1 \leqslant ND \leqslant NC$.

pour tout $1 \leqslant i \leqslant NC$ et $i \neq ND$:

VAL [i] : une valeur de concentration à l'équilibre pour le composé chimique TCO [i].

Résultat : valeur de la concentration à l'équilibre pour le composé chimique TCO [ND].

b. Fonction VANHOOFT

Arguments : - DHO : enthalpie libre de réaction.

- DSO : entropie de réaction.

pour tout $1 \leqslant i \leqslant NID$:

TID [i] : identificateur d'une variable intervenant dans la relation (càd 'T' ou 'K').

- ND : numéro de la variable pour laquelle on doit déterminer la valeur.

pour tout $1 \leqslant i \leqslant NID$, $i \neq ND$:

VAL [i] : valeur de la variable TID[I].

Précondition : $1 \leqslant ND \leqslant NID$.

Résultat : valeur de la variable TID [ND].

c. Procédure DVGW

Argument : - V : concentration à l'équilibre d'un composé chimique.

Résultat : - VMIN : la valeur minimale que prendra V.
- VMAX : la valeur maximale que prendra V.

Postcondition : - VMAX > VMIN.
- VMIN, VMAX ≠ 0.
- CMIN ≤ VMIN, VMAX ≤ CMAX.

où CMIN : la concentration la plus faible que peut prendre théoriquement V.

CMAX : la concentration la plus forte que peut prendre théoriquement V.

CMIN et CMAX peuvent être déterminés théoriquement à partir des valeurs de solubilité des composés chimiques.

Toutefois, nous n'avons pas réalisé ce calcul actuellement.

d. Procédure DVVH

Argument : - V = 'T' ou 'K'.

Résultat : - VMIN = la valeur minimale que prendra V.
- VMAX = la valeur maximale que prendra V.

Postcondition : VMAX > VMIN.

---

3. Algorithmes

FONCTION GULWAAGE (K, NR, NC, TCO, TCS, ND, VAL)

```
DEBUT
    SI ND ⩾ NR + 1  ALORS
        CR ← 1;
        CP ← 1;
        POUR J ← 1 JUSQU'A NR FAIRE
              CR ← CR * VAL [J]^TCS [J] ;
        POUR J ← NR + 1 JUSQU'A NC FAIRE
              SI J ≠ ND  ALORS
                   CP ← CP * VAL [J]^TCS [J] ;
        SI CP ≠ 0 ALORS
             GULWAAGE ← (K * CR / CP)^(1/TCS[ND])
        SINON GULWAAGE ← 0
    SINON
        CR ← 1;
        CP ← 1;
        POUR J ← 1 JUSQU'A NR FAIRE
             SI J ≠ ND ALORS
                  CR ← CR * VAL [J]^TCS [J] ;
        POUR J ← NR + 1 JUSQU'A NC FAIRE
```

$$CP \leftarrow CP * VAL[J]^{TCS[J]};$$

SI CR * K ≠ 0 ALORS

$$GULWAAGE \leftarrow \left(\frac{CP}{CR * K}\right)^{1/TCS[ND]}$$

SINON GULWAAGE ← 0;

FIN

FONCTION VANHOOFT (DHO, DGO, TID, NID, VD, VAL)

```
DEBUT
    SI VD = 'K' ALORS
       I ← NUMERO ('T', TID, NID);
       DGO ← DHO - (VAL [ I ] * DGO);
       VANHOOFT ← EXP (-DGO/VAL [ I ] )
    SINON   (*PAS DE SENS*)
       I ← NUMERO ('K', TID, NID);
       VANHOOFT ← -DHO/(LN(VAL [ I ] )-DSO);
FIN
```

PROCEDURE DVGW (V, VMIN, VMAX)

```
DEBUT
    REPETER
          ECRIRE ('VALEUR MINIMALE POUR', V);
          REPETER LIREEL (VMIN) JUSQU'A VMIN ≠ 0;
          ECRIRE ('VALEUR MAXIMALE POUR', V);
          REPETER LIREEL (VMAX) JUSQU'A VMAX ≠ 0;
    JUSQU'A VMAX > VMIN;
FIN
```

C. Module "UTILITAIRE-GRAPHES"

1. Procédures de l'interface

a. Procedure DESSINGRAF

Arguments : - Pour tout $1 <= i <= N$: $X[i]$ = un réel représentant une valeur de VX.

$Y[i]$ = un réel représentant une valeur de VY pour une valeur $X[i]$ de VX.

- VX : l'identificateur de la variable en abscisse.
- VY : l'identificateur de la variable en ordonnée.
- PC1 : caractère indiquant s'il faut inclure l'origine du système d'axes (='1') ou déplacer l'origine du système d'axes vers le point le plus proche (='2')
- PC2 : caractère indiquant s'il ne faut accorder aucune importance au pas suivant X et Y (='1') ou non (='2').

Résultat : - le graphique VY/VX.

b. Procédure DROITEREG

Arguments : - AR, BR, CR : les paramètres de la droite de régression d'équation AR*X+BR*Y+CR = 0.

Résultat : la droite de régression dessinée à l'écran sur le graphique VX/VY.

2. Conception

Afin d'obtenir un graphique VY/VX représentatif de la variation de VY en fonction de la variation de VX, il faut être en mesure de calculer les coordonnées sur l'écran graphique (XC_i, YC_i) à partir des valeurs X_i de VX et des valeurs Y_i de VY.
(Y_i étant la valeur de VY déterminée pour une valeur X_i de VX).

Ce calcul est rendu possible grâce au système de 2 équations :

$$XC_i = A * X_i + B$$
$$YC_i = C * Y_i + D$$

où les paramètres A, B, C et D sont déterminés :

- en n'accordant aucune importance au pas suivant X et Y . On obtient de la sorte une répartition optimale des différents points sur l'écran mais on peut parfois avoir une représentation "faussée" de la variation VY/VX.
- en accordant de l'importance au pas suivant X et Y. On obtient de la sorte une moins bonne répartition des différents points sur l'écran mais dans tous les cas une représentation "exacte" de la variation VY/VX.

Si tous les points sont localisés dans un seul cadran, nous avons laissé le choix à l'utilisateur :

- d'inclure l'origine du système d'axes ((0, 0)) dans le graphique si ce n'était déjà pas fait;

- de déplacer l'origine du système d'axes vers le point le plus proche du point (0, 0).

Enfin, nous avons envisagé de pouvoir tracer sur le graphique VY/VX la droite de régression.

Pour ce faire, nous évaluons pour la valeur minimale et maximale de VX (XMIN et XMAX), la valeur estimée correspondante de VY ($YMIN_{est}$ et $YMAX_{est}$).
Il ne reste plus qu'à tracer le segment de droite entre le point (xmin, $ymin_{est}$) et le point (xmax, $ymax_{est}$).

3. Algorithmes

PROCEDURE DESSINGRAF (X, Y, N, VX, VY, PC1, PC2)

```
DEBUT
    YMIN ← MINIMUM (Y, N);
    YMAX ← MAXIMUM (Y, N);
    SI PC1 = '1' ALORS ORIGIMP (XMIN, XMAX, YMIN, YMAX,
                                OAX, OAY);
    SI PC1 = '2' ALORS ORIGPIMP (XMIN, XMAX, YMIN, YMAX,
                                 OAX, OAY);
    SI PC2 = '1' ALORS CAL1ABCD (XMIN, XMAX, YMIN, YMAX,
                                 A, B, C, D);
    SI PC2 = '2' ALORS CAL2ABCD (XMIN, XMAX, YMIN, YMAX,
                                 A, B, C, D);
    CALAXES (OAX, OAY, A, B, C, D, COAX, COAY);
    DESAXES (COAX, COAY);
```

```
    CALAFX (COAX, COAY, AX, AY);
    AFFICHER (AX, AY, VX);
    CALAFY (COAX, COAY, AX, AY);
    AFFICHER (AX, AY, VY);
    CALCOOR (X, Y, N, A, B, C, D, XC, YC);
    DESPOINTS (XC, YC, N);
FIN
```

PROCEDURE DROITEREG (AR, BR, CR)

```
DEBUT
    SI AR ≠ 0 ET BR ≠ 0 ALORS
        CIX ← ROUND (XMIN * A + B);
        CIY ← ROUND ((XMIN * (-AR/BR) + (-CR/BR))* C + D);
        CFX ← ROUND (XMAX * A + B);
        CFY ← ROUND ((XMAX * (-AR/BR) + (-CR/BR))* C + D)
    SINON SI AR = 0 ALORS
            CIY ← ROUND ((-CR/BR) * C + D);
            CIX ← MINX;
            CFY ← CIY;
            CFX ← MAXX
        SINON SI BR = 0 ALORS
                CIX ← ROUND ((-CR/AR) * A + B);
                CIY ← MINY;
                CFX ← CIX;
                CFY ← MAXY
FIN
```

PROCEDURE ORIGIMP (XMIN, XMAX, YMIN, YMAX, OAX, OAY)

Arguments : - XMIN : la plus petite valeur en abscisse.
- XMAX : la plus grande valeur en abscisse.
- YMIN : la plus petite valeur en ordonnée.
- YMAX : la plus grande valeur en ordonnée.

Résultats : - XMIN : la plus petite valeur en abscisse.
- XMAX : la plus grande valeur en abscisse.
- YMIN : la plus petite valeur en ordonnée.
- YMAX : la plus grande valeur en ordonnée.

Postconditions : - OAX : l'origine du système d'axes suivant X. (=0)
- OAY : l'origine du système d'axes suivant Y. (=0)

Règle : Si tous les points apparaissent dans le même cadran, inclure l'origine du système d'axes si pas déjà fait explicitement.

```
DEBUT
    OAX ← 0;
    OAY ← 0;
    SI XMIN * XMAX > 0 ALORS
          SI  XMIN > 0 ALORS
                XMIN ← 0
          SINON XMAX ← 0
```

```
    SI YMIN * YMAX > 0 ALORS
          SI YMIN > 0 ALORS
                YMIN ← 0
          SINON  YMAX ← 0;
FIN
```

PROCEDURE ORIGPIMP (XMIN, XMAX, YMIN, YMAX, OAX, OAY)

Arguments : - XMIN : la plus petite valeur en abscisse.
- XMAX : la plus grande valeur en abscisse.
- YMIN : la plus petite valeur en ordonnée.
- YMAX : la plus grande valeur en ordonnée.

Résultats : - OAX : l'origine du système d'axes suivant X.
- OAY : l'origine du système d'axes suivant Y.

Règle : Si tous les points apparaissent dans le même cadran, déplacer l'origine du système d'axes vers le point le plus proche de celle-ci.

```
DEBUT
    OAX ← 0;
    OAX ← 0;
    SI XMIN * XMAX > 0 ALORS
              SI XMIN < 0 ALORS OAX ← XMAX
              SINON OAX ← XMIN
    SI YMIN * YMAX > 0 ALORS
              SI YMIN < 0  ALORS OAY ← YMAX
              SINON   OAY ← YMIN;
FIN
```

PROCEDURE CAL1ABCD (XMIN, XMAX, YMIN, YMAX, A, B, C, D)

Arguments : - XMIN : la plus petite valeur en abscisse.
- XMAX : la plus grande valeur en abscisse.
- YMIN : la plus petite valeur en ordonnée.
- YMAX : la plus grande valeur en ordonnée.

Résultats : - A, B, C et D : les constantes du système d'équations

$$XC = A * X + B$$

$$YC = C * Y + D$$

Règles : le pas suivant X et Y n'a pas d'importance.

```
DEBUT
    SI (XMAX - XMIN) ≠ 0  ALORS
            A ← (AMAX - AMIN) / (XMAX - XMIN)
    SINON   A ← 0 ;
    B ← AMIN - (A * XMIN);
    SI (YMAX - YMIN) ≠ 0  ALORS
            C ← (BMAX - BMIN) / (YMAX - YMIN)
    SINON   C ← 0;
    D ← BMIN - (C * YMIN);
FIN
```

PROCEDURE CAL2ABCD (XMIN, XMAX, YMIN, YMAX, A, B, C, D)

Arguments : - XMIN : la plus petite valeur en abscisse.
- XMAX : la plus grande valeur en abscisse.
- YMIN : la plus petite valeur en ordonnée.
- YMAX : la plus grande valeur en ordonnée.

Résultats : - A, B, C et D : les constantes du système d'équations

$$X_C = A * X + B$$

$$Y_C = C * Y + D$$

Règles : le pas suivant X et Y a de l'importance.

```
DEBUT
    SI ABS (YMAX) ≥ ABS (YMIN)  ALORS
            PG VY ← ABS (YMAX)
    SINON   PG VY ← ABS (YMIN);
    SI ABS (XMAX) ≥ ABS (XMIN)  ALORS
            PG VX ← ABS (XMAX)
    SINON   PG VX ← ABS (XMIN);
    SI PG VY > PG VX  ALORS
       SI (YMAX - YMIN) ≠ 0  ALORS
              C ← IMAX - IMIN
                  YMAX - YMIN
       SINON C ← 0;
       A ← C;
       B ← IMIN - A * XMIN;
       D ← IMIN - C * YMIN;
```

```
SI PGVY < PGVX  ALORS
    SI (XMAX - XMIN) ≠ 0  ALORS
                A ← (IMAX - IMIN) / (XMAX - XMIN)
    SINON       A ← 0;
    C ← A;
    B ← IMIN - (A * XMIN);
    D ← IMIN - (C * YMIN);
SI PGVY = PGVX  ALORS
    SI (YMAX - YMIN) ≠ 0  ALORS
                C ← (IMAX - IMIN) / (YMAX - YMIN)
    SINON       C ← 0;
    SI (XMAX - XMIN) ≠ 0 ALORS
                A ← (IMAX - IMIN) / (XMAX - XMIN)
    SINON       A ← 0;
    B ← IMIN - (A * XMIN);
    D ← IMIN - (C * YMIN);
FIN
```

PROCEDURE CALAXES (OAX, OAY, A, B, C, D, COAX, COAY)

Arguments : - OAX : l'origine du système d'axes suivant X.

- OAY : l'origine du système d'axes suivant Y.
- A, B, C et D : les constantes du système d'équations

$$XC = A * X + B$$

$$YC = C * Y + D.$$

Résultats : - COAX : la coordonnée à l'écran suivant X
de l'origine du système d'axes.

- COAY : la coordonnée à l'écran suivant Y
de l'origine du système d'axes.

Postcondition : BMIN <=COAY <=BMAX
AMIN <=COAX <=AMAX.

```
DEBUT
    COAX ← ROUND (OAX * A + B);
    COAY ← ROUND (OAY * C + D)
FIN
```

PROCEDURE CALCOOR (X, Y, N, A, B, C, D, XC, YC)

Arguments : - Pour tout i : 1 <=i <=N : X [i] = un réel
Y [i] = un réel.

- A, B, C et D : les constantes du système d'équations

XC = A * X + B
YC = C * Y + D.

Résultats : - Pour tout i : 1 <=i <=N : XC [i] = un entier
YC [i] = un entier.

Postcondition : pour tout 1 <=i <=N : AMIN <=XC [i] <= AMAX
(ou XC [i] : la coordonnée à l'écran du point X [i] suivant X)
BMIN <=YC [i] <=BMAX

(ou YC [i] : la coordonnée à l'écran du point Y [i] suivant Y).

```
DEBUT
    POUR I ← 1 JUSQU'A N  FAIRE
        XC [I] ← ROUND (X [I] * A + B);
        YC [I] ← ROUND (Y [I] * C + D)
FIN
```

PROCEDURE CALAFX (COAX, COAY, VX, AX, AY)

Arguments : - COAX : la coordonnée à l'écran suivant X de l'origine du système d'axes.

- COAY : la coordonnée à l'écran suivant Y de l'origine du système d'axes.

- VX : l'identificateur de la variable en abscisse.

Precondition : - MINX <=COAX <=MAXX .
- MINY <=COAY <=MAXY .

Résultats : - AX : la coordonnée à l'écran suivant X où il faut afficher l'identificateur de la variable en abscisse.
- AY : la coordonnée à l'écran suivant Y où il faut afficher l'identificateur de la variable en abscisse.

```
DEBUT
    SI COAX/2 > AMAX/2  ALORS
        AX ← LC
    SINON  AX ← MAXX - (LC * LONGUEUR (VX)).
    SI COAY/2 > BMAX/2  ALORS
        AY ← COAY + HC
    SINON  AY ← COAY - HC
FIN
```

PROCEDURE CALAFY (COAX, COAY, VY, AX, AY)

Arguments : - COAX : la coordonnée à l'écran suivant X de l'origine du système d'axes.

- COAY : la coordonnée à l'écran suivant Y de l'origine du système d'axes.

- VY : identificateur de la variable en ordonnée.

Précondition : - MINX <=COAX <=MAXX.

- MINY <=COAY <=MAXY.

Résultats : - AX : la coordonnée à l'écran suivant X où il faut afficher l'identificateur de la variable en ordonnée.

- AY : la coordonnée à l'écran suivant Y où il faut afficher l'identificateur de la variable en ordonnée.

```
DEBUT
     SI COAY/2 > BMAX/2  ALORS
                AY ← HC
     SINON      AY ← MAXY - HC ;
     SI COAX/2 > AMAX/2  ALORS
                AX ← COAX - (LC * LONGUEUR (VY))
     SINON      AX ← COAX + LC
FIN
```

D. Module "UTILITAIRE-E/S"

1. Fonctions de l'interface

FONCTION LIRENTIER

Résultat : un entier.

Postcondition : abs (lirentier) ≤ maxint.

FONCTION LIREEL

Résultat : un réel.

Postcondition : abs (lireel) ≤ maxréel.

---

2. Définition du langage

<ENTIER> ::= <SIGNE> <ENTIER NON SIGNE> / <ENTIER NON SIGNE>

<ENTIER NON SIGNE> ::= <CHIFFRE> / <CHIFFRE> <ENTIER NON SIGNE>

<SIGNE> ::= +/-

<REEL> ::= <SIGNE> <REEL NON SIGNE> / <REEL NON SIGNE>

<REEL NON SIGNE> ::= <ENTIER NON SIGNE> /
. <ENTIER NON SIGNE> /
E <ENTIER> /
<ENTIER NON SIGNE> . <ENTIER NON SIGNE> /
<ENTIER NON SIGNE> E <ENTIER> /
. <ENTIER NON SIGNE> E <ENTIER> /
<ENTIER NON SIGNE> . <ENTIER NON SIGNE>
E <ENTIER>

3. Conception

L'utilisateur introduit une suite de caractères (mémorisée dans le tableau LIGNE) . On procède alors à l'analyse syntaxique de cette suite de caractères afin de déterminer si celle-ci repésente bien un entier (ou un réel) selon le langage défini. Pour ce faire, on parcourera le tableau LIGNE de gauche à droite, poscc indiquant la position de la tête de lecture, en reconstituant progressivement l'entier (ou le réel).
En cas d'erreurs, la nature et l'emplacement de la 1ère erreur rencontrée seront signalés à l'utilisateur.

---

4. Algorithmes

PROCEDURE LIREEL (REEL)

```
DEBUT
    REPETER
        LIRE (LIGNE); LGLGN ← LONGUEUR (LIGNE);
        ERR ← FALSE;
        POSCC ← 0; CC ← ''; TCC ← TBOL;
        INV ← 1;
        LR ← 0; LE ← 0; LI ← 0; LS ← 0;
        PIN ← 0; PRE ← 0; PEX ← 1;
        SI LIRE (CC, TCC) ALORS
            SI CC = '-' ALORS
                SI LIRE (CC, TCC) ALORS INV ← -1;
                                       LS ← 1;
```

```
                SI CC = '+' ALORS
                   SI LIRE (CC, TCC) ALORS INV ← 1;
                                          LS ← 1;
                SI TCC = CHIFFRE ALORS PARTIENTIER
                SINON SI CC = '.' ALORS PARTIEREEL
                      SINON SI CC = 'E' ALORS
                                PARTIEXPO ;
                                PIN ← 1
                            SINON ERREUR
          SINON ERREUR ;
          SI ERR = FALSE ALORS REEL ← (PIN + PRE)* PEX*INV
    JUSQU'A NOT ERR;
FIN
```

PROCEDURE PARTIEENTIER

```
DEBUT
    RECONSTITUE (PIN, LI);
    SI CC = '.' ALORS PARTIEREEL
    SINON  SI CC = 'E' ALORS PARTIE XPO
           SINON  SI (LI + LS) < LGLGN ALORS ERREUR;
FIN
```

PROCEDURE PARTIEREEL

```
DEBUT
    SI LIRE (CC, TCC) ALORS
       SI TCC = CHIFFRE ALORS
          RECONSTITUE (PARTREEL, LR);
```

```
            PRE ← PARTREEL/10^LR;
            LR ← LR + 1;
            SI CC = 'E' ALORS PARTIEXPO
            SINON SI (LI + LR + LS) < LGLGN ALORS ERREUR
         SINON ERREUR
      SINON ERREUR;
FIN
```

PROCEDURE PARTIEXPO

```
DEBUT
    SI LIRE (CC, TCC) ALORS
       SI TCC = CHIFFRE ALORS
          RECONSTITUE (PARTIEXP, LR);
          PEX ← 10^PARTIEXP;
          LE ← LE + 1;
          SI (LI + LR + LE + LS) < LGLGN ALORS ERREUR
       SINON
          SI CC = '-' ALORS
           SI LIRE(CC, TCC) ALORS
              SI TCC = CHIFFRE ALORS
                  RECONSTITUE (PARTIEXP, LR);
                  PEX ← 10^-PARTIEXP;
                  LE ← LE + 2;
                  SI (LI + LR + LE + LS) < LGLGN ALORS  ERREUR
              SINON ERREUR;
           SINON ERREUR;
        SINON ERREUR;
    SINON ERREUR;
FIN
```

PROCEDURE LIRENTIER(ENTIER)

```
DEBUT
    REPETER
          LIRE (LIGNE); LGLGN ← LONGUEUR (LIGNE);
          ERR ← FALSE;
          POSSCC ← 0; CC ←' '; TCC ← TBOL;
          INV ← 1;
          LENT ← 0;
          SI LIRE (CC, TCC) ALORS
             SI CC = '-' ALORS
                   SI LIRE (CC, TCC) ALORS INV ← -1;
                                          LENT ← 1;
             SI CC = '+' ALORS
                   SI LIRE (CC, TCC) ALORS INV ← 1;
                                          LENT ← 1;
             SI TCC = CHIFFRE ALORS
                RECONSTITUE (ENT, NBENT);
                LENT ← LENT + NBENT;
                SI LENT < LGLGN ALORS ERREUR
             SINON ERREUR
          SINON ERREUR;
          SI NOT ERR ALORS ENTIER ← ENT * INV;
    JUSQU'A NOT ERR;
FIN
```

PROCEDURE RECONSTITUE (INT, NBCHLU)

Argument : /.

Précondition : LIGNE [POSCC] = 1 chiffre.

Résultats : INT : valeur de l'entier reconstitué.

NBCHLU : longueur de l'entier reconstitué.

Postcondition : LIGNE [POSCC] ≠ 1 chiffre.

```
DEBUT
    INT ← 0;
    NBCHLU ← 1;
    REPETER
          INT ← INT * 10 + VAL (CC);
          SI LIRE (CC, TCC) AND (TCC = CHIFFRE ) ALORS
                                  NBCHLU ← NBCHLU + 1
    JUSQU'A  TCC ≠ CHIFFRE;
FIN
```

FONCTION LIRE (CC, TCC)

Argument : /.

Précondition : POSCC : position courante de la tête de lecture à partir de laquelle il faut lire le prochain caractère.

Résultat : vrai si POSCC < LONGUEUR (LIGNE)

POSCC = POSCC + 1

CC = le caractère lu.

TCC = le type du caractère lu.

faux sinon alors

POSCC = LONGUEUR(LIGNE).

CC = '#'.

TCC = TEOL.

```
DEBUT
    SI POSCC ≥ LGLGN ALORS
       CC ← '#';
       TCC ← TEOL
    SINON POSCC ← POSCC + 1;
          CC ← LIGNE [POSCC] ;
          TCC ← TYPECH(CC);
    LIRE ← CC ≠ '#';
FIN
```

E. Module "UTILITAIRE-REGLIN"

1. Procédure de l'interface

PROCEDURE REGLIN

Arguments : - pour tout $1 \leq i \leq N$: X [i] = une valeur de VX
Y [i] = une valeur observée de VY pour la valeur X [i] de VX.
- CHREG : choix de la regression souhaitée.

Résultat : - AR, BR, CR : les paramètres de la droite de régression AR * VX + BR * VY + CR=0 correspondant à CHREG.
- R : coefficient de correlation linéaire.

---

2. Conception

Nous avons retenu 3 façons d'obtenir la droite de régression :

a; la droite de régression de y en fonction de x obtenue en minimisant la somme des carrés des écarts parallèlement à l'axe des ordonnées Y.
Dans ce cas, on peut établir que :

$$y = \frac{\text{cov}\ (x,y)}{s_x^2} (x-\bar{x}) + \bar{y}$$

où $\text{cov}\ (x,y) = \sum_{i=1}^{n} x_i y_i - \frac{1}{n} \left(\sum_{i=1}^{n} x_i \right) \left(\sum_{i=1}^{n} y_i \right)$

$$s_x^2 = \sum_{i=1}^{n} x_i^2 - \frac{1}{n} \left(\sum_{i=1}^{n} x_i \right)^2$$

b; la droite de régression de x en fonction de y obtenue en minimisant la somme des carrés des écarts parallèlement à l'axe des abscisses X.

Dans ce cas, on peut établir que :

$$x = \frac{\text{cov}\,(x,y)}{s_y^2}\,(y-\bar{y}) + \bar{x}$$

où $s_y^2 = \sum_{i=1}^{n} y_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} y_i\right)^2$

c; la droite de régression orthogonale obtenue en minimisant la somme des carrés des écarts perpendiculairement à la droite recherchée elle-même.

Dans ce cas on peut établir que :

$$y = \bar{y} - \bar{x} * \left(\frac{2\,\text{cov}\,(x,y)}{s_x^2 - s_y^2 + \sqrt{(s_x^2 - s_y^2)^2 + 4\,\text{cov}^2(x,y)}} \right)$$

3. Algorithmes

PROCEDURE (X, Y, N, CHREG, AR, BR, CR, R)

```
DEBUT
    SX ← 0;
    POUR I ← 1 JUSQU'A N FAIRE
                SX ← SX + X[I];
    SY ← 0;
    POUR I ← 1 JUSQU'A N FAIRE
                SY ← SY + Y[I];
```

```
SXY ← 0;
POUR I ← 1 JUSQU'A N FAIRE
            SXY ← SXY + (X [I] * Y [I] );
SX2 ← 0;
POUR I ← 1 JUSQU'A N FAIRE
            SX2 ← SX2 + PE ( X [I] ,2);
SY2 ← 0;
POUR I ← 1 JUSQU'A N FAIRE
            SY2 ← SY2 + PE (Y [I] ,2);
CXY ← SXY - ((SX.SY)/N))/(N-1);
XM ← SX/N;
YM ← SY/N;
S2X ← (SX2 - (PE(SX,2)/N)) /(N-1);
S2Y ← (SY2 - (PE(SY,2)/N)) / (N-1);
SXX ← SQRT (S2X);
SYY ← SQRT (S2Y);
SI (SXX * SYY) ≠ 0 ALORS R ← CXY / (SXX * SYY);
                   SINON R ← 0;
SI R ≠ 0 ALORS
   SI CHREG = 1 ALORS
      AR ← CXY;
      BR ← -S2X;
      CR ← S2X * YM - CXY * XM;
   SI CHREG = 2 ALORS
      AR ← -S2Y;
      BR ← CXY;
      CR ← S2Y * XM - CXY * YM;
   SI CHREG = 3 ALORS
      AR ← 2 * CXY;
```

```
            BR ← - (S2X - S2Y + SQRT ((PE((S2X - S2Y),2)) +
                 (4 * PE (CXY,2))));
            CR ← ((-BR) * YM) - (AR * XM)
        SINON
          SI ABS (S2X) = 0 ALORS
            AR ← 1;
            BR ← 0;
            CR ← -XM;
          SI ABS (S2Y) = 0 ALORS
            AR ← 0;
            BR ← -1;
            CR ← YM;
FIN
```

F. Module "UTILITAIRE-ECHELS"

1. Fonction et procédures de l'interface

- PROCEDURE TRANSECHEL

Arguments : - VX : l'identificateur de la variable en abscisse.

- VY : l'identificateur de la variable en ordonnée.
- Pour tout i : 1 <=i <=N : X [i] = un réel représentant une valeur de VX.

  Y [i] = un réel représentant une valeur de VY pour une valeur X [i] de VX.

Résultats : - CHX : le numéro de la fonction de transformation appliquée suivant X.

- CHY : le numéro de la fonction de transformation appliquée suivant Y.
- VXC : CONCAT (f(CHX), VX)
- VYC : CONCAT (f(CHY), VY)
- pour tout 1 <=i <=N : XC [i] =f (CHX) (X[i])

  YC [i] = f (CHY) (Y[i])

  où f(CHOIX) = l'identificateur de la fonction de transformation de numéro CHOIX.

- FONCTION REALI

Arguments : - Pour tout i : 1 ≤ i ≤ N : T[i] = un réel.
- CHOIX : le numéro de la fonction de transformation.

Résultat : - VRAI si pour tout 1 ≤ i ≤ N : f(CHOIX) (T[i]) est possible où f (CHOIX) = l'identificateur de la fonction de transformation de numéro CHOIX.

- FAUX sinon.

- PROCEDURE TRANSFO

Arguments : - Pour tout i : 1 ≤ i ≤ N : A[i] = un réel.
- CHOIX : le numéro de la fonction de transformation.

Résultats : - pour tout 1 ≤ i ≤ n : AC[i] = f(CHOIX) (A[i]) où f(CHOIX) = l'identificateur de la fonction de transformation de numéro CHOIX.

- FONCTION TRANSPO

Arguments : - VA : l'identificateur de la variable.
- CHOIX : le numéro de la fonction de transformation.

Résultats : - VAC : concat (f(CHOIX), VA)
où f(CHOIX) = l'identificateur de la fonction de transformation de numéro CHOIX.

---

2. Algorithmes

PROCEDURE TRANSECHEL (X, Y, N, VX, VY, XC, YC, VXC VYC, CHX, CHY)

```
DEBUT
    OKX ← FALSE;
    REPETER
            MENU ('X');     (*PRESENTATION DES FONCTIONS
                        DISPONIBLES SUIVANT X*)
            LIRENTIER (CHX);
            SI REALI (CHX, X, N) ALORS
                    TRANSFO (CHX, X, N, XC);
                    TRANSPO (CHX, VX, VXC);
                    OKX ← TRUE;
    JUSQU'A OKX;
    OKY ← FALSE;
```

```
    REPETER
        MENU ('Y')  (*PRESENTATION DES FONCTIONS DISPONI-
                     BLES SUIVANT Y*)
        LIRENTIER (CHY);
        SI REALI (CHY, Y, N) ALORS
              TRANSFO (CHY, Y, N, YC);
              TRANSPO (CHY, VY, VYC);
              OKY ← TRUE;
    JUSQU'A OKY;
FIN
```

FONCTION REALI (CHOIX , T, N)

```
DEBUT
    SUIVANT (CHOIX) FAIRE
        POUR I ← 1 JUSQU'A N FAIRE
            SI "T [I] EST OK POUR F(CHOIX) (T [I] )" ALORS
                REALI ← TRUE;
            SINON   REALI ← FALSE;
FIN
```

PROCEDURE TRANSFO (CHOIX, A, N, AC)

```
DEBUT
    SUIVANT(CHOIX) FAIRE
        POUR I ← 1 JUSQU'A N FAIRE
            AC [I] ← F(CHOIX) (A [I] );
FIN
```

PROCEDURE TRANSFO (CHOIX, VA, VAC)

```
DEBUT
    SUIVANT (CHOIX) FAIRE
        VAC ← CONCAT (F(CHOIX), '(', VA, ')');
FIN
```

III. MODULE "COORDINATEUR"

1° Programme permettant de retrouver inductivement la loi de GULDBERG - WAAGE

```
DEBUT
    ECRIRE ('EQUATION :');
    TANT QUE PAS EQUACHIM (NC, NR, TCO, TCS, TCH, TYRE) FAIRE
         ECRIRE ('EQUATION :');
    ECRIRE ('TEMPERATURE :');
    LIREEL (TEMP);
    ECRIRE (NP);
    LIRENTIER (NP);
    SI GETKE (TCO, TCS, NR, NC, TEMP, K)  ALORS
                PINFER (NC, NP, TCO, TEQU)
                POUR TOUT I ← 1 JUSQU'A (2^(NC-1)-1) FAIRE
                        ECRIRE ('EQUATION', I, ':', TEQU[I])
FIN
```

2° Programme permettant de retrouver inductivement la loi de VAN 'T HOFF

```
DEBUT
    ECRIRE ('EQUATION :')
    TANT QUE PAS EQUACHIM (NC, NR, TCO, TCS, TCH, TYRE) FAIRE
           ECRIRE ('EQUATION :');
    ECRIRE ('NP :');
    LIREEL (NP);
```

```
   SI GETDHO (TCO, TCS, NR, NC, DHO) ET GETDSO (TCO, TCS
                                        NR, NC, DSO)
   ALORS      TID [1] ← 'T';
              TID [2] ← 'K' ;
              NID ← 2;
              PINFER (NID, ND, TID, TEQU);
              ECRIRE (TEQU [1] );
FIN
```

3° Programme permettant la visualisation graphique de la variation des concentrations à l'équilibre en fonction de la température

```
DEBUT
    FIN ← FALSE;
    REPETER
          ECRIRE ('INTRODUIRE UNE EQUATION');
          TANT QUE PAS EQUACHIM (NC,NR,TCO,TCS,TCH,TYRE) FAIRE
               ECRIRE ('INTRODUIRE UNE EQUATION')
          SI GETDHO (TCO,TCS,NR,NC,DHO) ET GETDSO (TCO,
                                   TCS,NR,NC,DSO)
          ALORS  ECRIRE (DHO, DSO);
                 ECRIRE ('NP:');
                 LIRENTIER (NP);
                 TANT QUE NP > LMTAB FAIRE
                        ECRIRE ('NP:');
                        LIRENTIER (NP);
                 VX ← 'T';
```

```
                VARDOM (NP, VX, TMIN, TMAX, TSTEP);
                GETCONCINI (TCO, NR, CI);
                KFT (NP, TMIN, TSTEP, DHO, DSO, TK, TT);
                POUR I ← 1 JUSQU'A NP FAIRE
                      GETCONCEQUIL (TCS, CI, NR, NC,TK[I],CE);
                      POUR J ← JUSQU'A NC FAIRE
                            CT [I, J] ← CE [J];
                FINI ← FALSE;
                REPETER
                      ECRIRE ('VY:');
                      TANT QUE PAS COMPCHIM (VY) FAIRE
                            ECRIRE ('VY:');
                      SI EXVATA (VY,TCO,NC,NY) ALORS
                            FINI ← TRUE
                JUSQU'A FINI ;
                POUR I ← 1 JUSQU'A NP FAIRE
                      CC [I] ← CT [I, NY];
                DESSINGRAF (CC,TT,NP,VX,VY,PC1,PC2)
          SINON FIN ← TRUE
    JUSQU'A FIN
FIN
```

PROCEDURE KTT (N, TMIN, TSTEP, DHO, DSO, TK, TT)

Arguments : - DHO : enthalpie libre de réaction
- DSO : entropie de réaction.
- N : le nombre de points.
- TMIN : valeur minimale de la température.
- TMAX : valeur maximale de la température.

Résultats : pour tout $1 \leq i \leq N$:

TT [i] = TMIN + (I-1) * TSTEP.
TK [i] = valeur de K à la température TT [i].

```
DEBUT
    POUR I ← 1 JUSQU'A N FAIRE
       TT [I] ← TMIN + ((I-1) * TSTEP),
       DGO ← DHO -(TT [I] * DSO) ;
       TK [I] ← EXP ((-DGO)/TT [I] );
FIN
```

PROCEDURE GETCONCINI (TCO, NR, CI)

Arguments : NR : nombre de réactifs
pour tout $1 \leq i \leq NR$:

TCO [i] = un réactif.

Résultats : pour tout $1 \leq i \leq NR$:

CI [i] = concentration initiale pour le réactif TCO [i]

```
DEBUT
    POUR I ← 1 JUSQU'A NR FAIRE
        ECRIRE ('CONCENTRATION INITIALE DU REACTIF',TCO [I]);
        LIREEL (CI [I] )
FIN
```

---

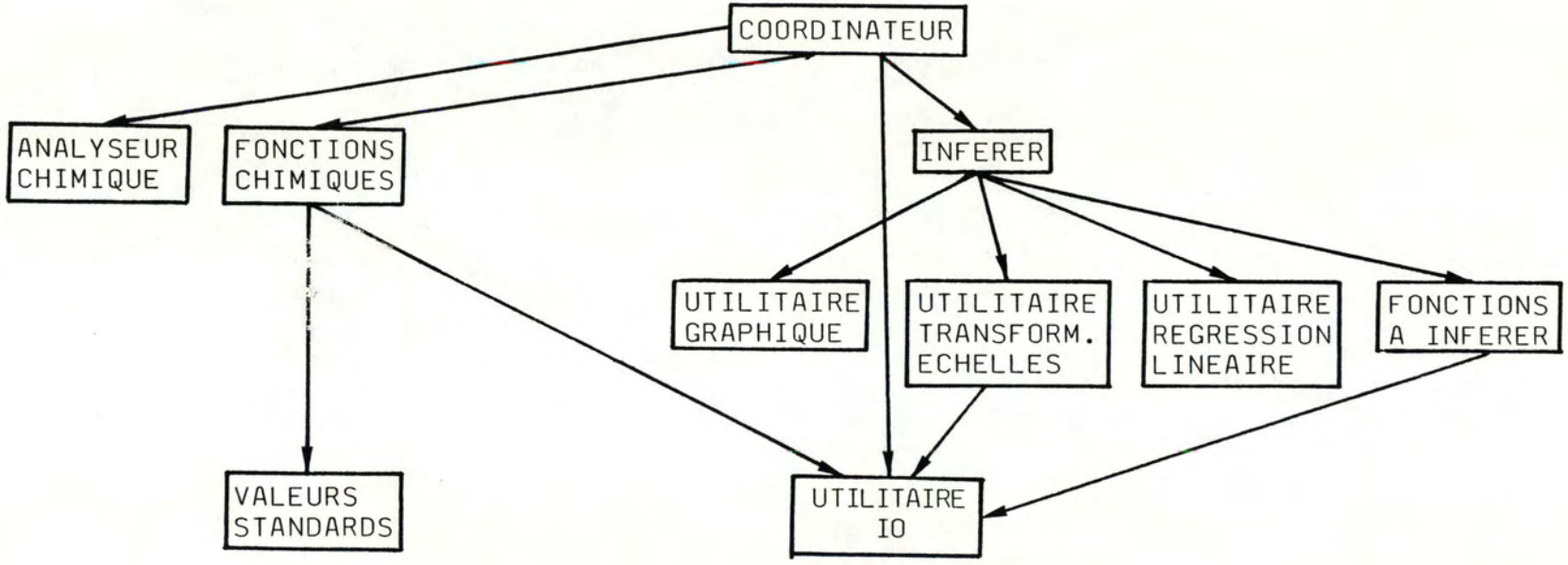
2. ARCHITECTURE PHYSIQUE

1. Découpe en modules physiques

A chaque module logique correspondra un module physique de façon à refleter le mieux possible la découpe en modules logiques réalisée lors de l'analyse organique.

2. Architecture physique proprement dite

Voir schéma page 209.


COORDINATEUR
ANALYSEUR CHIMIQUE
FONCTIONS CHIMIQUES
INFERER
UTILITAIRE GRAPHIQUE
UTILITAIRE TRANSFORM. ECHELLES
UTILITAIRE REGRESSION LINEAIRE
FONCTIONS A INFERER
VALEURS STANDARDS
UTILITAIRE IO

3. Codage des modules et intégration

Nous avons choisi d'implémenter notre didactiel en pascal sur le micro-ordinateur APPLE-II.

Avant d'en arriver au coeur du sujet, il faut commencer par parler du fonctionnement de l'APPLE-II.

Pour exécuter un programme de grosse taille, il est nécessaire de le découper en plusieurs "parties" qui peuvent chacune "tenir" en mémoire centrale. Deux possibilités sont offertes par le pascal UCSD disponible sur l'APPLE-II.

1° Déclarer quelques grosses procédures (ou fonctions) comme "SEGMENTS". Cela a pour effet de ne charger ces dernières en mémoire centrale que lorsqu'elles sont appelées effectivement.

2° Diviser le programme en différentes "parties" appelées "UNITS". Chaque "UNIT" constitue une entité compilable séparément. En donnant une indication appropriée au compilateur (option "NO LOAD"), il est possible de ne charger en mémoire centrale chaque "UNIT" que lorsque l'on en a besoin.

Nous avons choisi d'utiliser les "UNITS". Les raisons principales :

1° une fois la "UNIT" définie, on peut l'utiliser dans n'importe quel programme. Il suffit de le préciser au début du programme (ou d'une UNIT mère)

par l'instruction : uses 'nom de la UNIT'.

2° l'utilisation de "SEGMENTS" impose nécessairement d'avoir un gros programme principal. En effet, il n'est pas permis de déclarer des "SEGMENTS" à l'intérieur de "UNITS".

3° le nombre de "SEGMENTS" que l'on peut déclarer est limité et ceux-ci doivent impérativement apparaître en tête du programme principal.

L'utilisation des "UNITS" impose évidement une structure spéciale.
Chaque"UNIT" comprend :

- un nom qui l'identifie;
- une partie interface qui contient la partie de la "UNIT" qui est "visible de l'extérieur" càd :
  - *les éventuels appels à d'autres "UNITS"
  - * les éventuelles déclarations de données
  - * la ou les procédures (ou fonctions) qui peuvent être appelées par les programmes qui utilisent cette "UNIT".
- une partie implémentation qui comprend le texte des procédures (ou fonctions).

Nous allons maintenant reprendre le code de chacune des "UNITS" identifiées. Chaque "UNIT" correspond en fait à un module physique.

```
(*$S++ *)

unit VARGLOB;

interface

const lmtab=10;
      lmtabfn=10;
      lmtstring=15;
      lmtab2=5;
      lmtab3=4;
      lmtab4=3;
      lmtab5=4;

type tstring=string[lmtstring];
     tab=array[1..lmtab] of real;
     tab2=array[1..lmtab2,1..lmtab2] of real;
     tab3=array[1..lmtab3] of real;
     tab4=array[1..lmtab4] of tstring;
     tab5=array[1..lmtab5] of integer;
     tabn=array[1..lmtabfn] of integer;
     tabf=array[1..lmtabfn] of tstring;
     typere=(directe,equilibree);

function PE(x:real;y:integer):real;
function EXVATA(v:tstring;var tva:tabf;nv:integer;var num:integer):boolean;
function VARIA(num:integer;var tva:tabf):tstring;
function NUMERO(v:tstring;var tva:tabf;nv:integer):integer;

(*************************************************************************************************
)

implementation

function EXVATA;

var i:integer;
begin
exvata:=false;
for i:=1 to nv do
  if v=tva[i] then
  begin exvata:=true;
        num:=i;
  end;
end;

(*************************************************************************************************
)

function VARIA;
begin
varia:=tva[num];
end;

(*************************************************************************************************)
```

```
function NUMERO;

var i:integer;

begin
for i:=1 to nv do
 if tva[i]=v then numero:=i;
end;

{*****************************************************************************}


function PE;

var z:real;

begin if y < 0 then
            begin
             x:=1/x;
             y:=abs(y);
            end;
       z:=1;
       while y > 0 do
        begin
          y:=y-1;
          z:=z*x;
        end;
       pe:=z;
end;

{*****************************************************************************}

begin
end.
```

```
(*$S++ *)
unit ANALCHIMIE;

interface

uses (*$U #5:VARGLOB.CODE *) varglob;

function COMPCHIM(var comp:tstring):boolean;
function EQUACHIM(var nc,nr:integer;var tco:tabf;var tcs,tch:tabn;
                  var tyre:typere):boolean;

(*************************************************************************)

implementation

const lmligne=80;

type tligne=packed array[1..lmligne] of char;
     atome=(zz,h,he,li,be,b,c,n,o,f,ne,na,mg,al,si,p,s,cl,ar,k,ca,sc,ti,v,cr,
            mn,fe,co,ni,cu,zn,ga,ge,as,se,br,kr,rb,sr,y,zr,nb,mo,tc,ru,rh,pd,
            ag,cd,sn,sb,te,i,xe,cs,ba,la,hf,ta,w,re,os,ir,pt,au,hg,tl,pb,
            bi,po,at,rn,fr,ra,ac,ce,pr,nd,pm,sm,eu,gd,tb,dy,ho,er,tm,yb,lu,
            th,pa,u);
     tabat=array[atome] of integer;
     typec=(tbol,teol,min,maj,num);

var  poscc,lglgn:integer;
     ligne:tligne;
     typecc:typec;
     cc:char;

procedure CONCATENA(var comp:tstring;ch:integer);
var st:tstring;

begin
str(abs(ch),st);
if ch > 0 then comp:=concat(comp,'[',st,'+]');
if ch < 0 then comp:=concat(comp,'[',st,'-]');
end;

(*************************************************************************)

procedure ADD(var a,b:tabat);
var at:atome;

begin
for at:=h to u do a[at]:= a[at]+b[at]
end;

(*************************************************************************)

procedure MUL(var a:tabat;cs:integer);
var at:atome;

begin
for at:=h to u do a[at]:=a[at]*cs
end;

(*************************************************************************)
```

```
procedure MISEZERO(var a:tabat);
var at:atome;

begin
for at:=h to u do a[at]:=0;
end;

{*************************************************************************************}

procedure ERREUR(numerr:integer);

begin gotoxy(0,22);
      writeln('ERREUR : ',numerr,'  CARACTERE : ',poscc);
      writeln('<RETURN>');
end;

{*************************************************************************************}

function GETNEXT(var cc:char;var typecc:typec):boolean;

   function TYPECH(ch:char):typec;

   begin
   if (ord(ch)>96)and(ord(ch)<123) then typech:=min;
   if (ord(ch)>64)and(ord(ch)<91)  then typech:=maj;
   if (ord(ch)>48)and(ord(ch)<58) then typech:=num;
   end;

{*************************************************************************************}

   begin (*GETNEXT*)
   if poscc>=lglgn then
     begin
     cc:='#';typecc:=teol;
     end
   else
       begin
       poscc:=poscc+1;cc:=ligne[poscc];typecc:=typech(cc);
       end;
   getnext:=cc<>'#';
   end;

{*************************************************************************************}

function GETMULT(var mult:integer):boolean;

    function VAL(c:char):integer;
    begin
      val:=ord(c)-48;
    end;

{*************************************************************************************}

    begin
    if cc='0' then begin
                    mult:=0;getmult:=false;
                   end
    else
       if typecc=num then
          begin
           mult:=val(cc);
           while getnext(cc,typecc) and ((typecc=num) or (cc='0')) do
```

```
            mult:=10*mult+val(cc);
           getmult:=true;
          end
       else
          begin
           mult:=1;getmult:=true;
          end;
    end;

{*************************************************************************************}

function GETCHARGE(var charge:integer):boolean;
var signe:char;
    mult:integer;
    b:boolean;

     function GETSIGNE(var signe:char):boolean;

      begin
       getsigne:=true;
       if cc='+' then signe:='+'
       else
         if cc='-' then signe:='-'
         else getsigne:=false;
       end;

{*************************************************************************************}

    begin (*GETCHARGE*)
     getcharge:=true;
     charge:=0;
     if cc='[' then
      if getnext(cc,typecc) and getmult(mult) then
       begin
        if getsigne(signe) then
         begin if signe='+' then charge:=mult
               else charge:=-mult;
               if getnext(cc,typecc) and (cc=']') then
                begin b:=getnext(cc,typecc);getcharge:=true;end
               else begin getcharge:=false;erreur(6);end;
         end
        else begin getcharge:=false;erreur(5);end;
       end
      else begin getcharge:=false;erreur(4); end;
    end;

{*************************************************************************************}

function GETATOME(var simple:string;var atom:atome):boolean;
var compt:integer;

  function CONV(ss:string):atome;

    procedure SUITE3;
    begin
    if ss='Bi' then conv:=bi else if ss='Po' then conv:=po else
    if ss='At' then conv:=at else if ss='Rn' then conv:=rn else
    if ss='Fr' then conv:=fr else if ss='Ra' then conv:=ra else
    if ss='Ac' then conv:=ac else if ss='Ce' then conv:=ce else
    if ss='Pr' then conv:=pr else if ss='Nd' then conv:=nd else
    if ss='Pm' then conv:=pm else if ss='Sm' then conv:=sm else
    if ss='Eu' then conv:=eu else if ss='Gd' then conv:=gd else
```

```
if ss='Tb' then conv:=tb else if ss='Dy' then conv:=dy else
if ss='Ho' then conv:=ho else if ss='Er' then conv:=er else
if ss='Tm' then conv:=tm else if ss='Yb' then conv:=yb else
if ss='Lu' then conv:=lu else if ss='Th' then conv:=th else
if ss='U' then conv:=u;
end;

procedure SUITE2;
begin
if ss='Sn' then conv:=sn else if ss='Sb' then conv:=sb else
if ss='Te' then conv:=te else if ss='I' then conv:=i else
if ss='Xe' then conv:=xe else if ss='Cs' then conv:=cs else
if ss='Ba' then conv:=ba else if ss='La' then conv:=la else
if ss='Hf' then conv:=hf else if ss='Ta' then conv:=ta else
if ss='W' then conv:=w else if ss='Re' then conv:=re else
if ss='Os' then conv:=os else if ss='Ir' then conv:=ir else
if ss='Pt' then conv:=pt else if ss='Au' then conv:=au else
if ss='Hg' then conv:=hg else if ss='Tl' then conv:=tl else
if ss='Pb' then conv:=pb else suite3;
end;

procedure SUITE1;
begin
if ss='Ni' then conv:=ni else if ss='Cu' then conv:=cu else
if ss='Zn' then conv:=zn else if ss='Ga' then conv:=ga else
if ss='Ge' then conv:=ge else if ss='As' then conv:=as else
if ss='Se' then conv:=se else if ss='Br' then conv:=br else
if ss='Kr' then conv:=kr else if ss='Rb' then conv:=rb else
if ss='Sr' then conv:=sr else if ss='Y' then conv:=y else
if ss='Zr' then conv:=zr else if ss='Nb' then conv:=nb else
if ss='Mo' then conv:=mo else if ss='Tc' then conv:=tc else
if ss='Ru' then conv:=ru else if ss='Rh' then conv:=rh else
if ss='Pd' then conv:=pd else if ss='Ag' then conv:=ag else
if ss='Cd' then conv:=cd else suite2;
end;

begin (*CONV*)
conv:=zz;
if ss='H' then conv:=h else if ss='He' then conv:=he else
if ss='B' then conv:=b else if ss='C' then conv:=c else
if ss='N' then conv:=n else if ss='O' then conv:=o else
if ss='F' then conv:=f else if ss='Ne' then conv:=ne else
if ss='Na' then conv:=na else if ss='Mg' then conv:=mg else
if ss='Al' then conv:=al else if ss='Si' then conv:=si else
if ss='P' then conv:=p else if ss='S' then conv:=s else
if ss='Cl' then conv:=cl else if ss='Ar' then conv:=ar else
if ss='K' then conv:=k else if ss='Ca' then conv:=ca else
if ss='Sc' then conv:=sc else if ss='Ti' then conv:=ti else
if ss='V' then conv:=v else if ss='Cr' then conv:=cr else
if ss='Mn' then conv:=mn else if ss='Fe' then conv:=fe else
if ss='Co' then conv:=co else suite1;
end;

(*************************************************************************)

function EXISATOME(var simple:string;var ato:atome):boolean;
var at:atome;

begin exisatome:=false;
      ato:=conv(simple);
      for at:=h to u do
          if at=ato then exisatome:=true
```

```
  end;

{*****************************************************************************}

  begin
   getatome:=false;
   simple:='          ';
   if typecc=maj then
    begin
     simple[1]:=cc;
     compt:=1;
     while getnext(cc,typecc) and (typecc=min) do
      begin
       compt:=compt+1;
       simple[compt]:=cc;
      end;
    end;
   if simple<>'          ' then
    begin
     simple:=copy(simple,1,compt);
     if exisatome(simple,atom) then getatome:=true
     else erreur(2);
    end;
  end;

{*****************************************************************************}

function GETCOMPOSE(var composc:tstring;var nbatco:tabat):boolean;

var nbatli:tabat;
    simple:string;
    muli:integer;
    atom:atome;
    st,ligand:tstring;

  function GETLIGAND(var ligand:tstring;var nbatli:tabat):boolean;

  var st:tstring;
      simple:string;
      mult:integer;
      atom:atome;
      b:boolean;

  begin
   ligand:='';getligand:=false;misezero(nbatli);
   while getatome(simple,atom) and getmult(mult) do
     begin
         nbatli[atom]:=nbatli[atom]+mult;
         if mult=1 then ligand:=concat(ligand,simple)
         else begin str(mult,st);ligand:=concat(ligand,simple,st);end;
     end;
   if (length(ligand)<>0)and(mult<>0) then
    begin
    if (cc=')') then begin b:=getnext(cc,typecc);
                           getligand:=true;
                     end
                else erreur(3);
    end;
  end;

{*****************************************************************************}
```

```
    begin (*GETCOMPOSE*)
      compose:='';getcompose:=false;miseazero(nbatco);
      while getatome(simple,atom) and getmult(mult) do
       begin
          nbatco[atom]:=nbatco[atom]+mult;
          if mult=1 then compose:=concat(compose,simple)
          else
           begin str(mult,st);compose:=concat(compose,simple,st);end;
       end;
      if (length(compose)<>0) and (mult<>0) then
       begin
       if cc = '(' then
        begin
         if getnext(cc,typecc) and getligand(ligand,nbatli) and getmult(mult)
         then
          begin
           str(mult,st);
           compose:=concat(compose,'(',ligand,')',st);
           mul(nbatli,mult);
           add(nbatco,nbatli);
           getcompose:=true;
          end;
        end
       else getcompose:=true;
       end;
    end;

(*************************************************************************)

function LIRECHIMIE(var ligne:tligne;var lglgn:integer):boolean;
var i:integer;
    fin:boolean;
    x,y:integer;

  procedure SAISIE(var ligne:tligne;var lglgn:integer);
  var ch:char;

  function ACCEPTABLE(var car:char):boolean;

  begin
  acceptable:=(car in ['A'..'Z','a'..'z','0'..'9','+','-','<','>','(',')','=',
  ' ','.','[',']']) or (ord(car)=8) );
  end;

(*************************************************************************)

  function MEMORISABLE(var car:char):boolean;
  begin
  memorisable:=(car in ['A'..'Z','a'..'z','0'..'9','+','-','(',')','[',']','='
  ,'<','>',' ']);
  end;

(*************************************************************************)

  begin (*SAISIE*)
  repeat read(keyboard,ch) until acceptable(ch);
  if memorisable(ch) then
   begin
    lglgn:=lglgn+1;
    ligne[lglgn]:=ch;
    write(ch);
    x:=x+1;
```

```
      if lglgn >= lmligne then fin:=true;
     end
    else
     begin
      case ord(ch) of
      46:fin:=true;
      8:begin
        if lglgn>=1 then
         begin
          ligne[lglgn]:=chr(32);
          lglgn:=lglgn-1;
          x:=x-1;
          gotoxy(x,y);
          write(' ');
          gotoxy(x,y);
         end
        end
       end
      end;
     end;

(*****************************************************************************************)

  procedure ENLEVERBLANC(var ligne:tligne;var lglgn:integer);
  var k:integer;

  begin
  k:=0;
  for i:=1 to lglgn do
   begin
    if ligne[i]<>chr(32) then
     begin
      k:=k+1;
      if k<i then
       begin
       ligne[k]:=ligne[i];
       ligne[i]:=chr(32);
       end
     end;
   end;
  lglgn:=k
  end;

(*****************************************************************************************)

     begin (*LIRECHIMIE*)
     x:=0;y:=3;gotoxy(x,y);
     fin:=false;lglgn:=0;
     for i:=1 to lmligne do ligne[i]:=chr(32);
     lirechimie:=true;
     repeat saisie(ligne,lglgn) until fin;
     if lglgn=0 then
          begin
           gotoxy(0,22);
           erreur(12);
           lirechimie:=false;
          end
     else
          begin
           writeln;
           if lglgn=lmligne then
               begin
```

```
                gotoxy(0,22);
                erreur(13);
                lirechimie:=false;
               end
          else
               begin
               enleverblanc(ligne,lglgn);
               if lglgn=0 then
                    begin
                     gotoxy(0,22);
                     erreur(14);
                     lirechimie:=false;
                    end;
               end;
          end;
    end;

(*****************************************************************************)

function EQUACHIM;

var  nbatre,nbatpr:tabat;
     chre,chpr:integer;

function SYNTAXIQUE(var ligne:tligne;lglgn:integer;var nc,nr,chre,
chpr:integer;var tco:tabf;var tcs,tch:tabn;var tyre:typere;
var nbatre,nbatpr:tabat):boolean;

function GETEQUATION:boolean;
var np:integer;

function GETLISTEPRODUIT(var nclp,chlp:integer;var nbatlp:tabat):boolean;

var nbatpro:tabat;
    continue:boolean;

function GETPRODUIT(var co:tstring;var cs,ch:integer;var nbatpr:tabat):boolean;

  function EXISTDEJA(s:integer):boolean;
  var i:integer;

  begin existdeja:=false;
        i:=1;
        while i<=(s-1) do
        begin if (tco[i]=tco[s]) then
              existdeja:=true;
              i:=i+1
        end
  end;

(*****************************************************************************)

  begin (*PRODUIT*)
  getproduit:=false;
  if getnext(nc,typecc) then
   begin
   if not getmult(cs) then erreur(1) else
    if getcompose(co,nbatpr) then
     if getcharge(ch) then
      begin concatena(co,ch);
            if existdeja(nc) then erreur(7) else getproduit:=true;
      end;
```

```
  end
 end;

{*************************************************************************}

     begin (*LISTEPRODUITS*)
      getlisteproduit:=false;
      nclp:=0;chlp:=0;
      continue:=true;
      misezero(nbatlp);
      repeat
       nc:=nc+1;
       if getproduit(tco,tcs,tch,nbatpro) then
        begin
          nclp:=nclp+1;
          chlp:=chlp+tch[nc];
          mul(nbatpro,tcs[nc]);add(nbatlp,nbatpro);
          getlisteproduit:=true;
        end
       else
        begin
         continue:=false;
         getlisteproduit:=false;
        end
      until (continue=false) or (cc<>'+');
     end;

{*************************************************************************}

function GETYRE(var tyre:typere):boolean;

  begin
    getyre:=false;
    if cc='-' then
     begin
      if getnext(cc,typecc) and (cc='-') then
       if getnext(cc,typecc) and (cc='>') then
        begin
        tyre:=directe;
        getyre:=true;
        end
     end
    else
     begin
      if cc='<' then
       if getnext(cc,typecc) and(cc='=') then
        if getnext(cc,typecc) and (cc='=') then
         if getnext(cc,typecc) and (cc='>') then
          begin
          tyre:=equilibree;
          getyre:=true;
          end;
     end;
  end;

  {*************************************************************************}

  begin (*EQUATION*)
  getequation:=false;
   if getlisteproduit(nr,chre,nbatre) then
     begin
      if getyre(tyre) then
```

```
        begin
         if getlisteproduit(np,chpr,nbatpr) then getequation:=true;
        end
        else erreur(9);
      end
  end;

{*****************************************************************************}

  begin (*SYNTAXIQUE*)
   cc:=' ';typecc:=tbol;poscc:=0;
   syntaxique:=false;
   if getequation then syntaxique:=true;
  end;

{*****************************************************************************}

function SEMANTIQUE (chre,chpr:integer;var nbatre,nbatpr:tabat):boolean;

 function EQUILSTOECHIO(var nbatre,nbatpr:tabat):boolean;
   var at:atome;

   begin
   equilstoechio:=true;
   for at:=h to u do
   if nbatre[at]<>nbatpr[at] then equilstoechio:=false
   end;

{*****************************************************************************}

 function EQUILELEC (chre,chpr:integer):boolean;

   begin
   equilelec:=true;
   if chre<>chpr then equilelec:=false;
   end;

{*****************************************************************************}

   begin (*SEMANTIQUE*)
    semantique:=false;
    if equilstoechio(nbatre,nbatpr) then
     begin
      if equilelec(chre,chpr) then
       semantique:=true
      else erreur(8);
     end
    else erreur(9);
   end;

{*****************************************************************************}

begin (*EQUACHIM*)
equachim:=false;
nc:=0;
if lirechimie(ligne,lglgn) then
 if syntaxique(ligne,lglgn,nc,nr,chre,chpr,tco,tcs,tch,tyre,nbatre,nbatpr)
 then if semantique(chre,chpr,nbatre,nbatpr) then
   equachim:=true;
end;

{*****************************************************************************}
```

```
function ANALCOMP(var ligne:tligne;lglgn:integer;var comp:tstring):boolean;
var nbatcomp:tabat;
    chargecomp:integer;

begin cc:=' ';typecc:=tbol;poscc:=0;analcomp:=false;
      if getnext(cc,typecc) and getcompose(comp,nbatcomp) and
      getcharge(chargecomp) then begin analcomp:=true;
                                       concatena(comp,chargecomp);
                                 end;
end;

(*****************************************************************************)

function COMPCHIM:

begin
compchim:=false;
if lirechimie(ligne,lglgn) then
 if analcomp(ligne,lglgn,comp) then
  if length(comp) = lglgn then compchim:=true
  else erreur(11);
end;

(*****************************************************************************)

begin
end.
```

```
(*$S++*)

unit VALSTA;

interface

const f1='#5:FICHMS.DATA';
      f3='#5:FICHMC.DATA';
      f2='#5:AUXFICH.DATA';
      lmtabcomp=50;

type identifiant=string[15];
     tcharge=integer;
     attribut=record
                h0f:real;
                s0f:real;
              end;
     article= record
                id:identifiant;
                att:attribut;
              end;
     fichcomp=file of article;
     tabcomp=array[1..lmtabcomp] of article;

var f,af:fichcomp;
    t:tabcomp;

function EXISTMOLECULE(var f:fichcomp;sf:string ;var t:tabcomp;nbc:integer;
                       i:identifiant;var at:attribut):boolean;
function EXISMS(var f:fichcomp;sf:string;i:identifiant;var at:attribut):
                boolean;

function EXISMC(var t:tabcomp;var n:integer;i:identifiant;var at:attribut):
boolean;

procedure CONSFSEQ(var f:fichcomp;sf:string);

procedure CONSID(var f:fichcomp;sf:string;i:identifiant);

procedure ENREGIS(var f:fichcomp;sf:string;a:article);

procedure MODIFIC(var f:fichcomp;sf:string;a:article);

procedure SUPPRES(var f:fichcomp;sf:string;i:identifiant);

procedure CHARGER(var f:fichcomp;sf:string;var t:tabcomp);

function COMPTER(var f:fichcomp;sf:string):integer;

procedure TRI(var f:fichcomp;sf:string;nbart:integer);

(*************************************************************************)

implementation


var trouve:boolean;
```

```
procedure TRI;

const dim=100;
type tabcomp=array[1..dim] of article;
var ta:tabcomp;
    n:0..dim;

procedure LECTURE;

begin
reset(f,sf);
n:=0;
while not eof(f) do
 begin
  n:=n+1;
  ta[n]:=f^;
  get(f);
 end;
close(f,lock);
end;

(*************************************************************************)

procedure ECRITURE;

var i:1..dim;

begin
rewrite(f,sf);
for i:=1 to n do
 begin f^:=ta[i];
       put(f);
 end;
close(f,lock);
end;

(*************************************************************************)

procedure TSHELL(g,d:integer);

var p,i,j,k:integer;
    s:article;

begin
 p:=1;
 while p < ((d-g+1) div 9) do
  p:=p*3+1;
 repeat
  for k:=g to p do
   begin
    i:=k+p;
    if i <= d then
     repeat
      s:=ta[i];
      j:=i-p;
      while (j >= k+p) and (ta[j].id > s.id) do
       begin
        ta[j+p]:=ta[j];
        j:=j-p;
       end;
      if ta[k].id > s.id then
       begin
```

```
          j:=k-p;
          ta[k+p]:=ta[k];
         end;
        ta[j+p]:=s;
        i:=i+p
       until i > d;
     end;
    p:=p div 3
  until p=0;
end;

(*************************************************************************)

begin
(*$I-*)
reset(f,sf);
if ioresult <> 0 then
 begin
  writeln(sf,' PAS TROUVE');
  exit(program);
 END;
close(f,lock);
(*$I+*)
if nbart <= dim then
 begin
   lecture;
   tshell(1,n);
   ecriture
 end
else writeln('FICHIER TROP GRAND');
end;

(*************************************************************************)

function COMPTER;

var n:integer;

begin
(*$I-*)
reset(f,sf);
if ioresult <> 0 then
 begin
  writeln(sf,' PAS TROUVE');
  exit(program);
 END;
(*$I+*)
n:=0;
while not eof(f) do
 begin
  n:=n+1;
  get(f);
 end;
close(f,lock);
compter:=n;
end;

(*************************************************************************)
```

```
function EXISMS;

begin trouve:=false;
      (*$I-*)
      reset(f,sf);
      if ioresult <> 0 then
       begin
        writeln(sf,' PAS TROUVE');
        exit(program);
       END;
      (*$I+*)
      while (not eof(f)) and (not trouve) do
      begin if (i=f^.id) then
            begin at.hOf:=f^.att.hOf;
                  at.sOf:=f^.att.sOf;
                  trouve:=true
            end
            else get(f)
      end;
      if trouve then exisms:=true
                else exisms:=false;
      close(f,lock)
end;

(*****************************************************************************************)

function EXISMC;

var j:integer;

begin trouve:=false;
      j:=0;
      while (j<n) and (not trouve) do
      begin j:=j+1;
            if (i=t[j].id) then
               begin at.hOf:=t[j].att.hOf;
                     at.sOf:=t[j].att.sOf;
                     trouve:=true
               end
      end;
      if trouve then exismc:=true
                else exismc:=false;

end;

(*****************************************************************************************)

function EXISTMOLECULE;

begin
existmolecule:=true;
if (not exismc(t,nbc,i,at)) then
 if (not exisms(f,sf,i,at)) then existmolecule:=false;
end;

(*****************************************************************************************)
```

```
procedure TRANSFERT(var af:fichcomp;saf:string;var f:fichcomp;sf:string);
begin reset(af,saf);
      rewrite(f,sf);
      while not eof(af) do
      begin f^:=af^;
            get(af);
            put(f)
      end;
      close(af,lock);
      close(f,lock);
end;

(*****************************************************************************)

procedure MODIFIC;

begin (*$I-*)
      reset(af,f2);
      if ioresult <> 0 then
       begin rewrite(af,f2);
             close(af,lock);
       end
      else close(af,lock);
      (*$I+*)
      reset(f,sf);
      rewrite(af,f2);
      while not eof(f) do
      begin if (f^.id<>a.id) then
               begin af^:=f^;
                     put(af);
                     get(f);
               end
            else
               begin get(f);
                     af^:=a;
                     put(af)
               end
      end;
      close(f,lock);
      close(af,lock);
      transfert(af,f2,f,sf);
end;

(*****************************************************************************)

procedure SUPPRES;

begin (*$I-*)
      reset(af,f2);
      if ioresult <> 0 then
       begin rewrite(af,f2);
             close(af,lock);
       end
      else close(af,lock);
      (*$I+*)
      reset(f,sf);
      rewrite(af,f2);
      while not eof(f) do
      begin if (f^.id<>i) then
               begin af^:=f^;
                     put(af);
                     get(f);
```

```
                end
              else get(f);

        end;
        close(f,lock);
        close(af,lock);
        transfert(af,f2,f,sf);
end;

{*****************************************************************************}

procedure ENREGIS;

begin {$I-}
        reset(af,f2);
        if ioresult <> 0 then
         begin rewrite(af,f2);
               close(af,lock);
         end
        else close(af,lock);
        {$I+}
        reset(f,sf);
        rewrite(af,f2);
        while not eof(f) do
          begin af^:=f^;
                put(af);
                get(f);
          end;
        close(f,lock);
        af^:=a;
        put(af);
        close(af,lock);
        transfert(af,f2,f,sf);
        writeln;
end;

{*****************************************************************************}

procedure CONSFSEQ;

begin {$I-}
        reset(f,sf);
        if ioresult <> 0 then
         begin
          writeln(sf,' PAS TROUVE');
          exit(program);
         END;
        {$I+}
        if eof(f) then writeln('LE FICHIER NE COMPREND AUCUN COMPOSE')
        else
         begin
          while not eof(f) do
          begin writeln('COMPOSE : ',f^.id);
                writeln('HOF : ',f^.att.hOf);
                writeln('SOF : ',f^.att.sOf);
                writeln;
                get(f);
          end;
         end;
        close(f,lock);
end;
```

```
(*****************************************************************************)

procedure CONSID;

var at:attribut;

begin
if exisms(f,sf,i,at) then
  begin writeln('HOF : ',at.hof);
        writeln('SOF : ',at.sof);
        writeln;
  end
else writeln('CE COMPOSE N''EST PAS MEMORISE');
end;

(*****************************************************************************)

procedure CHARGER;

var n:integer;

begin
(*$I-*)
reset(f,sf);
if ioresult <> 0 then
 begin
  writeln(sf,' PAS TROUVE');
  exit(program);
 END;
(*$I+*)
n:=0;
while not eof(f) do
 begin
  n:=n+1;
  t[n]:=f^;
  get(f);
 end;
close(f,lock);
end;

(*****************************************************************************)

begin
end.
```

```
(*$S++ *)

unit CALCHIM1;

interface

uses transcendend,
     (*$U #5:VARGLOB.CODE *) varglob,
     (*$U #5:VALSTA.CODE *) valsta;

var nbcmc:integer;

function GETKE(var tc:tabf;var tcs:tabn;nr,nc:integer;
                                        var t,k:real):boolean;
function GETDGO(var tc:tabf;var tcs:tabn;var nr,nc:integer;
                                        var t,dgO:real):boolean;
function GETDHO(var tc:tabf;var tcs:tabn;nr,nc:integer;var dhO:real)
                                                    :boolean;
function GETDSO(var tc:tabf;var tcs:tabn;nr,nc:integer;var dsO:real)
                                                    :boolean;
function GETDGR (var tc:tabf;var tcs:tabn;nr:integer;var t,dgr:real)
                                                    :boolean;
function GETDHR(var tc:tabf;var tcs:tabn;nr:integer;var dhr:real)
                                                    :boolean;
function GETDSR(var tc:tabf;var tcs:tabn;nr:integer;var dsr:real)
                                                    :boolean;
function GETDGP (var tc:tabf;var tcs:tabn;nr,nc:integer;var t,dgp
                                                  :real):boolean;
function GETDHP(var tc:tabf;var tcs:tabn;nr,nc:integer;var dhp:real)
                                                    :boolean;
function GETDSP(var tc:tabf;var tcs:tabn;nr,nc:integer;var dsp:real)
                                                    :boolean;
(*************************************************************************)


implementation



function GETDHR;

var j:integer;
    i:identifiant;
    continue:boolean;
    at:attribut;

begin
getdhr:=true;
continue:=true;
dhr:=0;
j:=1;
while ((j<=nr) and continue) do
  begin i:=tc[j];
        if existmolecule(f,fl,t,nbcmc,i,at) then
          begin  dhr:=dhr+(tcs[j]*at.h0f);
                 j:=j+1;
          end
        else
          begin  writeln('NOUS NE DISPOSONS PAS DE LA VALEUR DE L''ENTHALPIE');
                 writeln('DU COMPOSE ',i);
                 getdhr:=false;
```

```
                    continue:=false;
            end;
  end;
end;

(*****************************************************************************)

function GETDSR;

var j:integer;
    i:identifiant;
    continue:boolean;
    at:attribut;

begin
getdsr:=true;
continue:=true;
dsr:=0;
j:=1;
while ((j<=nr) and continue) do
  begin i:=tc[j];
        if existmolecule(f,f1,t,nbcmc,i,at) then
          begin  dsr:=dsr+(tcs[j]*at.s0f);
                 j:=j+1;
          end
        else
          begin  writeln('NOUS NE DISPOSONS PAS DE LA VALEUR DE L''ENTROPIE');
                 writeln('DU COMPOSE ',i);
                 getdsr:=false;
                 continue:=false;
          end;
  end;
end;

(*****************************************************************************)

function GETDHP;

var j:integer;
    i:identifiant;
    continue:boolean;
    at:attribut;

begin
getdhp:=true;
continue:=true;
dhp:=0;
j:=nr+1;
while ((j<=nc) and continue) do
  begin i:=tc[j];
        if existmolecule(f,f1,t,nbcmc,i,at) then
          begin  dhp:=dhp+(tcs[j]*at.h0f);
                 j:=j+1;
          end
        else
          begin  writeln('NOUS NE DISPOSONS PAS DE LA VALEUR DE L''ENTHALPIE');
                 writeln('DU COMPOSE ',i);
                 getdhp:=false;
                 continue:=false;
          end;
  end;
end;
```

```
(*****************************************************************************)

function GETDSP;

var j:integer;
    i:identifiant;
    continue:boolean;
    at:attribut;

begin
getdsp:=true;
continue:=true;
dsp:=0;
j:=nr+1;
while ((j<=nc) and continue) do
  begin i:=tc[j];
        if existmolecule(f,f1,t,nbcmc,i,at) then
          begin  dsp:=dsp+(tcs[j]*at.s0f);
                 j:=j+1;
          end
        else
          begin  writeln('NOUS NE DISPOSONS PAS DE LA VALEUR DE L''ENTROPIE');
                 writeln('DU COMPOSE ',i);
                 getdsp:=false;
                 continue:=false;
          end;
  end;
end;

(*****************************************************************************)

function GETDH0;
var dhr,dhp:real;

begin
getdh0:=true;
if getdhr(tc,tcs,nr,dhr) and getdhp(tc,tcs,nr,nc,dhp) then
   dh0:=dhp-dhr
else getdh0:=false;
end;

(*****************************************************************************)

function GETDS0;
var dsr,dsp:real;

begin
getds0:=true;
if getdsr(tc,tcs,nr,dsr) and getdsp(tc,tcs,nr,nc,dsp) then
   ds0:=dsp-dsr
else getds0:=false;
end;

(*****************************************************************************)
```

```
function GETDGR;
var dhr,dsr:real;

begin
  getdgr:=true;
  if getdhr(tc,tcs,nr,dhr) and getdsr(tc,tcs,nr,dsr) then
     dgr:=dhr-(t*dsr)
  else getdgr:=false;
end;

(*************************************************************************************)

function GETDGP;
var dhp,dsp:real;

begin
  getdgp:=true;
  if getdhp(tc,tcs,nr,nc,dhp) and getdsp(tc,tcs,nr,nc,dsp) then
     dgp:=dhp-(t*dsp)
  else getdgp:=false;
end;

(*************************************************************************************)

function GETDGO;
var dhO,dsO:real;

begin
getdgO:=false;
if getdhO(tc,tcs,nr,nc,dhO) and
 getdsO(tc,tcs,nr,nc,dsO) then
  begin
   getdgO:=true;
   dgO:=dhO-(t*dsO);
  end;
end;

(*************************************************************************************)

function GETKE;

const r=1;
var dgO:real;

begin
getke:=false;
if getdgO(tc,tcs,nr,nc,t,dgO) then
 begin
   k:=exp((-dgO)/(r*t));
   getke:=true;
 end;
end;

(*************************************************************************************)

begin
nbcmc:=compter(f,f3);
charger(f,f3,t);
end.
```

```
(*$S+ *)

unit CALCHIM2;

interface

uses transcendend,
     (*$U #5:VARGLOB.CODE *) varglob,
     (*$U #5:UTILIO.CODE *) utilio;

function VALDGO(dhO,dsO,t:real):real;

function VALKE(dgO,t:real):real;

procedure GETCONCEQUIL(var tcs:tabn;var ci:tab;nr,nc:integer;var t:real
                       ;var ce:tab);
procedure GETCONCINI(var tco:tabf;nr:integer;var ci:tab);

(*****************************************************************************)


implementation

function VALDGO;

begin
valdgO:=dhO-(t*dsO);
end;

(*****************************************************************************)

function VALKE;
const r=1;

begin
valke:=exp((-dgO)/(r*t));
end;

(*****************************************************************************)

procedure GETCONCEQUIL;

var xa,xb,rac:real;
    i:integer;


function SCSP :integer;

var s:integer;

begin
s:=O;
for i:=nr+1 to nc do s:=s+tcs[i];
scsp:=s;
end;

(*****************************************************************************)
```

```
function PPCSP : real;

var f:real;

begin
f:=1;
for i:=nr+1 to nc do f:=f*pe(tcs[i],tcs[i]);
ppcsp:=f;
end;

(*****************************************************************************)

function G (x:real):real;

var f:real;

begin
f:=1;
for i:=i to nr do f:=f*pe(ci[i]-(tcs[i]*x),tcs[i]);
g:=k*f;
end;

(*****************************************************************************)

function H (x:real):real;

var f:real;

begin
f:=pe(x,scsp);
h:=f*ppcsp;
end;

(*****************************************************************************)

function F (x:real):real;

begin
f:=g(x)-h(x);
end;

(*****************************************************************************)

procedure DICHO(xa,xb:real;var rac:real);

const predicho=0.000001;

var z,x,w:real;
    s:boolean;

begin  s:=f(xa)<0;
       repeat x:=(xa+xb)/2.0;
              z:=f(x);
              if (z<0)=s then xa:=x else xb:=x;
              w:=abs(xa-xb)
       until w < predicho;
       rac:=x;
end;

(*****************************************************************************)
```

```
procedure VALINTERVAL(var tcs:tabn;var ci:tab;nr:integer;var xa,xb:real);

var i:integer;
    val:tab;

function MINIMUM(var a:tab;n:integer):real;

var i:integer;
    min:real;

begin
min:=a[1];
i:=1;
while i < n do
 begin
  i:=i+1;
  if a[i] < min then min:=a[i];
 end;
minimum:=min;
end;

begin
for i:=1 to nr do
 val[i]:=ci[i]/tcs[i];
xa:=0;
xb:=minimum(val,nr);
end;

(*************************************************************************************)

begin(*GETCONCEQUIL*)
valinterval(tcs,ci,nr,xa,xb);
if f(xa)*f(xb) < 0 then
  begin
   dicho(xa,xb,rac);
   for i:=1 to nr do ce[i]:=(ci[i]-(tcs[i]*rac));
   for i:=nr+1 to nc do ce[i]:=(tcs[i]*rac);
  end
else writeln('LE CHOIX DE L''INTERVALLE EST MAUVAIS');
end;

(*************************************************************************************)

procedure GETCONCINI;

var i:integer;

begin
for i:=1 to nr do
 begin
  write('CONCENTRATION INITIALE EN ',tco[i],' : ');
  lireel(ci[i]);
 end;
end;

(*************************************************************************************)

begin
end.
```

```
(*$S+,N+ *)

unit INFERER;

interface

uses turtlegraphics,transcendend,
     (*$U #5:VARGLOB.CODE *) varglob,
     (*$U #5:UTILIO.CODE *) utilio,
     (*$U #5:UTILREG.CODE *) utilreg,
     (*$U #5:UTILECHEL.CODE *) utilechel,
     (*$U #5:FONCINF.CODE *) foncinf,
     (*$U #5:UTILDES.CODE *) utildes;

const lmtabeq=7;

type tabeq=array[1..lmtabeq] of string;

var nc,nr,nid:integer;
    tcs,tch:tabn;
    tco,tid:tabf;
    k,dh0,ds0:real;

procedure PINFER(nv,n:integer;var tva:tabf;chf:char;var teq:tabeq);

(*****************************************************************************)

implementation

var i:integer;

procedure PINFER;

const lmtpor=5;

type tabtpor=array[1..lmtpor] of tstring;

var reussi:boolean;
    reponse:char;
    snumeq,vx,vy:tstring;
    cx,cy,val:tab;
    no:tab5;
    ex,ey,nx,ny,numeq:integer;
    pente,origine:real;
    num3,num4,num5:integer;
    cmin,cstep:tab3;
    chx,chy:integer;
    tp,tor:tabtpor;


function EXVACR(var no:tab5;num:integer):boolean;

begin
exvacr:=false;
 if no[num] = 1 then exvacr:=true;
end;

(*****************************************************************************)
```

```
procedure RETIRER(num:integer;var no:tab5);
begin
no[num]:=0;
end;

{*****************************************************************************}

procedure REMETTRE(num:integer;var no:tab5);
begin
no[num]:=1;
end;

{*****************************************************************************}

procedure INITCRIBLE(var no:tab5;nv:integer);

var i:integer;

begin
for i:=1 to nv do no[i]:=1;
end;

{*****************************************************************************}

function GETVAR(var no:tab5;nv:integer):integer;

var i:integer;
    trouve:boolean;

begin i:=0;
      trouve:=false;
      while (i<nv) and (not trouve) do
        begin
         i:=i+1;
         if exvacr(no,i) then
           begin
            getvar:=i;
            trouve:=true;
           end;
        end;
end;

{*****************************************************************************}

procedure CHOIX(c:char;var tva:tabf;nv:integer;var no:tab5;var v:string;
                var nuv:integer);

var fin:boolean;

begin
fin:=false;
repeat
write(c,' : ');
readln(v);
if exvata(v,tva,nv,nuv) then
 if exvacr(no,nuv) then fin:=true
until fin;
end;

{*****************************************************************************}
```

```
procedure DOMVAR(var tid:tabf;var no:tab5;nid,np:integer;var cmin,cstep:tab5);

var i:integer;
    cmax:real;

begin
for i:=1 to nid do
 if exvacr(no,i) then
  begin
   case chf of
    '1':dvgulwaage(tid[i],cmin[i],cmax);
    '2':dvvanhooft(tid[i],cmin[i],cmax);
   end;
   cstep[i]:=cmax-cmin[i]/(np-1);
  end;
end;

(*****************************************************************************)

procedure PENTORIG(a,b,c:real;var p,o:real);

begin
if (a<>0) and (b<>0) then
 begin p:=(-a)/b;
       o:=(-c)/b;
 end;
if (a=0) and (b<>0) then
 begin p:=0;
       o:=(-c)/b;
 end;
writeln('PENTE : ',p);
writeln('COORDONNEE A L''ORIGINE : ',o);
end;

(*****************************************************************************)


function EXISTDROITE(var x,y:tab;n:integer;var vx,vy:tstring;var p,co:real)
                     :boolean;

const presup=0.99;
      preinf=0.05;
      resol=0.01;

var cc:real;
    ar,br,cr:real;
    pc1,pc2:char;

begin
(*$R TURTLEGRAPHICS*)
existdroite:=false;
writeln('[1]:ORIGINE A DE L''IMPORTANCE');
writeln('[2]:ORIGINE N''A PAS D''IMPORTANCE');
repeat read(keyboard,pc1) until (pc1 in ['1','2']);
writeln(pc1);
writeln('[1]:PAS DIFFERENTS');
writeln('[2]:PAS IDENTIQUES');
repeat read(keyboard,pc2) until (pc2 in ['1','2']);
writeln(pc2);
dessingraf(x,y,n,vx,vy,pc1,pc2,resol);
readln;
rl(x,y,n,2,ar,br,cr,cc);
```

```
pentorig(ar,br,cr,p,co);
droitereg(ar,br,cr);
readln;
textmode;
if (abs(cc)>presup) or (abs(cc)<preinf) then existdroite:=true;
end;

{*************************************************************************}

procedure SHOW(var x,y:tab;n:integer;var vx,vy:tstring);

var i:integer;

begin
for i:=1 to n do
 writeln(x,': ',x[i],'  ',y,': ',y[i]);
end;

{*************************************************************************}


function TROUVERDROITE(var x,y:tab;n:integer;var vx,vy:tstring;var p,co:real;
                       var chx,chy,ex,ey:integer):boolean;

var fini:boolean;
    xc,yc:tab;
    vxc,vyc:tstring;
    reponse:char;

begin
{**R TRANSCEND/END *}
trouverdroite:=false;
fini:=false;
xc:=x;yc:=y;vxc:=vx;vyc:=vy;chx:=1;chy:=1;
repeat
 if existdroite(xc,yc,n,vxc,vyc,p,co) then
  begin
   fini:=true;
   trouverdroite:=true;
  end
 else
  begin
   write('DESIREZ VOUS MODIFIER LES ECHELLES : ');
   readln(reponse);
   if reponse='O' then transechel(x,y,n,vx,vy,xc,yc,vxc,vyc,chx,chy,ex,ey)
   else fini:=true;
  end
until fini;
vx:=vxc;
vy:=vyc;
end;

{*************************************************************************}

procedure EQUATION(var vx,vy,p,o:tstring;var expeq:string);

begin
expeq:=' ';
expeq:=concat(vy,'=',p,'*',vx,'+',o);
end;
```

```
(*****************************************************************************)

procedure CALCUL((*VAR VAL:TAB;*)nx,ny:integer;var x,y:tab;n:integer);
var i:integer;

function VALY((*VAR VAL:TAB;*)ny:integer):real;

begin
case ch1 of
'1':valy:=gulwaage(nc,nr,ny,k,tco,tcs,val);
'2':valy:=vanhooft(tid,nid,ny,dh0,ds0,val);
end;
end;

(*****************************************************************************)

begin
(*$R TRANSCENDEND,FONCINF *)
for i:=1 to n do
 begin
  val[nx]:=cmin[nx]+((i-1)*cstep[nx]);
  x[i]:=val[nx];
  y[i]:=valy((*VAL,*)ny);
 end;
end;

(*****************************************************************************)


procedure CONVERTIR(var r:real;var s:string);

var pe,pr,pi:integer;
    se,sr,ss,si:string;

begin
pi:=0;pr:=0;pe:=0;
if r < 0 then ss:='-'else ss:=' ';
r:=abs(r);
if r <> 0 then
 begin
  if r > maxint then
   begin while r > maxint do
         begin
          r:=r/10;
          pe:=pe+1;
         end;
   end
  else
   begin if r < (maxint/10) then
         begin while r < (maxint/10) do
               begin
                r:=r*10;
                pe:=pe-1;
               end;
         end;
   end;
 end;
pi:=trunc(r);
str(pi,si);
pr:=trunc((r-pi)*10000);
str(pr,sr);
str(pe,se);
```

```
s:=concat(ss,si,'.',sr,'E',se);
end;

(*****************************************************************************)

function TEST2(var x,y:tab;n:integer;var vy,vx:tstring):boolean;

var spente,sorigine:string;

begin
test2:=false;
numeq:=numeq+1;
if trouverdroite(x,y,n,vx,vy,pente,origine,chx,chy,ex,ey) then
 begin
  convertir(pente,spente);
  convertir(origine,sorigine);
  equation(vx,vy,spente,sorigine,teq[numeq]);
  writeln(teq[numeq]);
  test2:=true;
 end;
end;

(*****************************************************************************)

function GETMEM2(var t1:tab;n:integer;var vy,vx:tstring;var cx,cy:tab):boolean;
var i:integer;

begin
getmem2:=true;
nx:=numero(vx,tva,nv);
for i:=1 to n do
 begin
  cx[i]:=cmin[nx]+((i-1)*cstep[nx]);
  cy[i]:=t1[i];
 end;
show(cx,cy,n,vx,vy);
end;

(*****************************************************************************)

function TEST3(var p,o:tstring;var l1,m1:tab;n:integer):boolean;
var vxc,vxcm:tstring;
    nxc:integer;

begin
test3:=false;
nxc:=getvar(no,nv);
vxc:=tva[nxc];
vxcm:=vxc;
retirer(nxc,no);
writeln('ETUDE DE ',p,'/',vxc);
if getmem2(l1,n,p,vxc,cx,cy) then
begin
 if test2(cx,cy,n,p,vxc) then
 begin
  writeln('ETUDE DE ',o,'/',vxcm);
  if getmem2(m1,n,o,vxcm,cx,cy) then
  begin
   if test2(cx,cy,n,o,vxcm) then test3:=true;
  end;
 end;
end;
```

```
remettre(nvc,no);
end;

{*****************************************************************************}

function GETMEM3(var t2:tab2;n:integer;var vy,vx:tstring;var ll,mi:tab1:boolean;
var i,j:integer;
    continue:boolean;

begin
getmem3:=false;
i:=i;
nx:=numero(vx,tva,nv);
for i:=1 to n do
 begin
  cx[j]:=cmin[nx]+((j-1)*cstep[nx]);
  if nx=num4 then
   cy[j]:=t2[j,i]
  else
   cy[j]:=t2[i,j];
 end;
val[num3]:=cmin[num3];
writeln('[',tva[num3],'] = ',val[num3]);
show(cx,cy,n,vx,vy);
if trouverdroite(cx,cy,n,vx,vy,ll[i],mi[1],,chx,chy,ex,ey) then
 begin
  continue:=true;
  while continue and (i<n) do
   begin
    i:=i+1;
    val[num3]:=cmin[num3]+((i-1)*cstep[num3]);
    j:=0;
    while continue and (j<n) do
     begin
      j:=j+1;
      cx[j]:=cmin[nx]+((j-1)*cstep[nx]);
      if nx=num4 then
       cy[j]:=t2[j,i]
      else
       cy[j]:=t2[i,j];
     end;
     if reali(chx,cx,n) and reali(chy,cy,n) then
      begin
       transfo(chx,ex,cx,n,cx);
       transfo(chy,ey,cy,n,cy);
       writeln('[',tva[num4],'] = ',val[num4]);
       show(cx,cy,n,vx,vy);
       if existdroite(cx,cy,n,vx,vy,ll[i],mi[1]) then
       else continue:=false;
      end
     else continue:=false;
   end;
 end
 else continue:=false;
if continue then getmem3:=true;
end;

{*****************************************************************************}
```

```
function TEST4(var p,o:tstring;var l2,m2:tab2;n:integer):boolean;

var li,mi:tab;
    vxc,vxcm:tstring;
    nxc:integer;

begin
test4:=false;
writeln('CHOIX DE LA VARIABLE EN ABSCISSE');
choix('X',tva,nv,no,vxc,nxc);
vxcm:=vxc;
retirer(nxc,no);
if nxc=num4 then num3:=num5 else num3:=num4;
writeln('ETUDE DE ',p,'/',vxc,' AVEC ',tva[num3],' CONSTANTE');
if getmem3(l2,n,p,vxc,li,mi) then
 begin
  numeq:=numeq+1;
  str(numeq,snumeq);
  tp[numeq]:=concat('P',snumeq);
  tor[numeq]:=concat('O',snumeq);
  equation(vxc,p,tp[numeq],tor[numeq],teq[numeq]);
  writeln(teq[numeq]);
  if test3(tp[numeq],tor[numeq],li,mi,n) then
   begin
    writeln('ETUDE DE ',o,'/',vxcm,' AVEC ',tva[num3],' CONSTANTE');
    if getmem3(m2,n,o,vxcm,li,mi) then
     begin
      numeq:=numeq+1;
      str(numeq,snumeq);
      tp[numeq]:=concat('P',snumeq);
      tor[numeq]:=concat('O',snumeq);
      equation(vxcm,o,tp[numeq],tor[numeq],teq[numeq]);
      writeln(teq[numeq]);
      if test3(tp[numeq],tor[numeq],li,mi,n) then test4:=true;
     end;
   end;
 end;
remettre(nxc,no);
end;

(*************************************************************************************)

procedure CHVAO;

begin
writeln('CHOIX DE LA VARIABLE EN ORDONNEE');
choix('Y',tva,nv,no,vy,ny);
retirer(ny,no);
case nv of
2:begin
   nx:=getvar(no,nv);
   vx:=tva[nx];
  end;
3,4:begin
     writeln('CHOIX DE LA VARIABLE EN ABSCISSE');
     choix('X',tva,nv,no,vx,nx);
    end;
end;
writeln('DOMAINE DE VARIATION');
domvar(tva,no,nv,n,cmin,cstep);
retirer(nx,no);
end;
```

```
(*****************************************************************************)

procedure VAR2;

function GETVAR2:boolean;
begin
writeln('ETUDE DE ',vy,'/ ',vx);
getvar2:=true;
calcul((*VAL,*)nx,ny,cx,cy,n);
show(cx,cy,n,vx,vy);
end;

begin
chvao;
if getvar2 then
 begin
  numeq:=0;
  if test2(cx,cy,n,vy,vx) then reussi:=true;
 end;
end;

(*****************************************************************************)

procedure VAR3;
var j:integer;
    lal,mul:tab;

function GETVAR3:boolean;
var continue:boolean;

begin
getvar3:=false;
j:=1;
num3:=getvar(no,nv);
writeln('ETUDE DE ',vy,'/',vx,' AVEC ',tval[num3],' CONSTANTE');
val[num3]:=cmin[num3];
writeln('EXPERIENCE ',j);
writeln('[',tval[num3],'] = ',val[num3]);
calcul((*VAL,*)nx,ny,cx,cy,n);
show(cx,cy,n,vx,vy);
if trouverdroite(cx,cy,n,vx,vy,lal[j],mul[j],chx,chy,ex,ey) then
 begin
  continue:=true;
  while continue and (j<n) do
   begin
    j:=j+1;
    val[num3]:=cmin[num3]+((j-1)*cstep[num3]);
    writeln('EXPERIENCE ',j);
    writeln('[',tval[num3],'] = ',val[num3]);
    calcul((*VAL,*)nx,ny,cx,cy,n);
    if reali(chx,cx,n) and reali(chy,cy,n) then
     begin
      transfo(chx,ex,cx,n,cx);
      transfo(chy,ey,cy,n,cy);
      show(cx,cy,n,vx,vy);
      if existdroite(cx,cy,n,vx,vy,pente,origine) then
       begin
        lal[j]:=pente;
        mul[j]:=origine;
       end
```

```
      else continue:=false;
     end
    else continue:=false;
   end;
  end
 else continue:=false;
if continue then getvar3:=true;
end;

(*************************************************************************)

begin
chvao;
if getvar3 then
 begin
  numeq:=1;
  tp[numeq]:='P1';tor[numeq]:='O1';
  equation(vx,vy,tp[numeq],tor[numeq],teq[numeq]);
  writeln(teq[numeq]);
  if test3(tp[numeq],tor[numeq],la1,mu1,n) then  reussi:=true;
 end;
end;

(*************************************************************************)

procedure VAR4;
var i,k,l:integer;
    la2,mu2:tab2;

function GETVAR4:boolean;
var continue:boolean;

begin
getvar4:=false;
i:=1;l:=1;
num4:=getvar(no,nv);
retirer(num4,no);
val[num4]:=cmin[num4];
k:=1;
num5:=getvar(no,nv);
remettre(num4,no);
writeln('ETUDE DE ',vy,'/',vx,' AVEC ',tva[num4],' ET ',tva[num5],' CONSTANTES')
;
val[num5]:=cmin[num5];
writeln('EXPERIENCE ',l);
writeln('[',tva[num4],'] = ',val[num4]);
writeln('[',tva[num5],'] = ',val[num5]);
calcul((*VAL,*)nx,ny,cx,cy,n);
show(cx,cy,n,vx,vy);
if trouverdroite(cx,cy,n,vx,vy,la2[j,k],mu2[j,k],chx,chy,ex,ey) then
 begin
  continue:=true;
  while continue and (j<=n) do
   begin
    while continue and (k<n) do
     begin
      k:=k+1;
      l:=l+1;
      writeln('EXPERIENCE ',l);
      val[num5]:=cmin[num5]+((k-1)*cstep[num5]);
      writeln('[',tva[num4],'] = ',val[num4]);
      writeln('[',tva[num5],'] = ',val[num5]);
```

```
        calcul((*VAL,*)nx,ny,cx,cy,n);
        if reali(chx,cx,n) and reali(chy,cy,n) then
         begin
          transfo(chx,ex,cx,n,cx);
          transfo(chy,ey,cy,n,cy);
          show(cx,cy,n,vx,vy);
          if existdroite(cx,cy,n,vx,vy,pente,origine) then
           begin
            la2[j,k]:=pente;
            mu2[j,k]:=origine;
           end
          else continue:=false;
         end
        else continue:=false;
       end;
       k:=0;
       j:=j+1;
       val[num4]:=cmin[num4]+((j-1)*cstep[num4]);
      end;
     end
   else continue:=false;
if continue then getvar4:=true;
end;

{*************************************************************************}

begin
chvao;
if getvar4 then
 begin
  numeq:=1;
  tp[numeq]:='P1';tor[numeq]:='01';
  equation(vx,vy,tp[numeq],tor[numeq],teq[numeq]);
  writeln(teq[numeq]);
  if test4(tp[numeq],tor[numeq],la2,mu2,n) then  reussi:=true;
 end;
end;

{*************************************************************************}

begin
reussi:=false;
repeat
initcrible(no,nv);
case nv of
 2:var2;
 3:var3;
 4:var4;
 5:writeln('ERREUR');
end;
if not reussi then
 begin
  write('ON CONTINUE : ');
  readln(reponse);
  if reponse='N' then reussi:=true;
 end
until reussi;
end;

{*************************************************************************}

begin
end
```

```
(*$S+,N+ *)

unit FONCINF;

interface

uses transcendend,
     (*$U #5:VARGLOB.CODE *) varglob,
     (*$U #5:UTILIO.CODE *) utilio;

function GULWAAGE (nc,nr,nd:integer;k:real;var tco:tabi;var tcs:tabn;
                   var val:tab):real;
procedure DVGULWAAGE(var v:tstring;var:vmin,vmax:real);
function VANHOOFT (var tid:tabi;nid,nd:integer;dh0,ds0:real;
                   var val:tab):real;
procedure DVVANHOOFT(var v:tstring;var vmin,vmax:real);

(*****************************************************************************)


implementation


function VANHOOFT;

var dg0:real;
    vd:tstring;

begin
vd:=tid[nd];
if vd='K' then
 begin
  dg0:=dh0-(val[nd]*ds0);
  vanhooft:=exp((-dg0)/val[nd]);
 end
else
 begin
  writeln('ERREUR');
  exit(program);
 END;
end;

(*****************************************************************************)

procedure DVVANHOOFT;

begin
repeat
 writeln('VALEUR MINIMALE POUR : ',v);
 lireel(vmin);
 writeln('VALEUR MAXIMALE POUR : ',v);
 lireel(vmax)
until vmax > vmin;
end;

(*****************************************************************************)
```

```
function GULWAAGE;

var cr,cp:real;
    j:integer;

begin
(*$R TRANSCENDEND *)
if (nd >= (nr+1))  then
   begin
    cr:=1;cp:=1;
    for j:=1 to nr do cr:=cr*pe(val[j],tcs[j]);
    for j:=nr+1 to nc do
        if (j<>nd) then  cp:=cp*pe(val[j],tcs[j]);
    if cp <> 0 then gulwaage:= exp((1/tcs[nd])*ln((k*cr)/cp)) else gulwaage:=0;
   end
else
   begin
    if (nd <= nr)  then
        begin
         cr:=1;cp:=1;
         for j:=1 to nr do
           if j<>nd then  cr:=cr*pe(val[j],tcs[j]);
         for j:=nr+1 to nc do cp:=cp*pe(val[j],tcs[j]);
         if cr*k <> 0 then gulwaage:=exp((1/tcs[nd]))*ln(cp/(cr*k))
         else gulwaage:=0;
        end;
   end;
end;

(*************************************************************************************)

procedure DVGULWAAGE;

begin
repeat
 writeln('VALEUR MINIMALE POUR : ',v);
 repeat lireel(vmin) until vmin > 0;
 writeln('VALEUR MAXIMALE POUR : ',v);
 repeat lireel(vmax) until vmax > 0
until vmax > vmin;

(*************************************************************************************)

begin
end.
```

```
(*$S+,N+ *)

unit UTILDES;

interface

uses turtlegraphics,
     (*$U #5:VARGLOB.CODE *) varglob;

procedure DESSINGRAF(var x,y:tab;n:integer;var vx,vy:tstring;pc1,pc2:char;
                     resol:real);
procedure DESSINDROITE(cix,ciy,cfx,cfy:integer);
procedure AFFICHER(ax,ay:integer;s:tstring);
procedure DPOINTS(var xc,yc:tabn;n:integer);
procedure DROITEREG(ar,br,cr:real);

{*****************************************************************************}

implementation

const maxx=279;
      maxy=191;
      minx=0;
      miny=0;
      amax=260;
      bmax=160;
      amin=0;
      bmin=0;
      imax=160;
      imin=0;
      xd=10;
      yd=20;
      gp=3;

var xmin,xmax,ymin,ymax,a,b,c,d:real;

procedure DESSINDROITE;

begin
(*$R TURTLEGRAPHICS *)
pencolor(none);
moveto(cix,ciy);
pencolor(white);
moveto(cfx,cfy);
end;


procedure DROITEREG;

var cix,ciy,cfx,cfy:integer;

begin
if (ar<>0) and (br<>0) then
 begin
  cix:=round(xmin*a+b);
  ciy:=round((xmin*(-ar/br)+(-cr/br))*c+d);
  cfx:=round(xmax*a+b);
  cfy:=round((xmax*(-ar/br)+(-cr/br))*c+d);
 end
else
```

```
  begin
   if ar=0 then
    begin
     ciy:=round((-cr/br)*c+d);
     cix:=minx;
     cfy:=ciy;
     cfx:=maxx;
    end;
   if br=0 then
    begin
     cix:=round((-cr/ar)*a+b);
     ciy:=miny;
     cfx:=cix;
     cfy:=maxy;
    end;
  end;
dessindroite(cix,ciy,cfx,cfy);
end;

{*****************************************************************************}

procedure DPOINTS;

 var i:integer;
     point:array[1..gp,1..gp] of boolean;

    procedure INIPOINT;

    var i,j:integer;

    begin
     for i:=1 to gp do
      for j:=1 to gp do
        point[i,j]:=true;
    end;

{*****************************************************************************}

      begin
       (*$R TURTLEGRAPHICS *)
       inipoint;
       for i:= 1 to n do
       drawblock(point,2,0,0,gp,gp,xc[i]-1,yc[i]-1,13);
      end;

{*****************************************************************************}

procedure AFFICHER;

 begin
 (*$R TURTLEGRAPHICS *)
 pencolor(none);
 moveto(ax,ay);
 wstring(s);
 end;

{*****************************************************************************}
```

```
procedure DESSINGRAF;

var     ax,ay,coax,coay:integer;
        oax,oay:real;
        re:char;
        xc,yc:tabn;
        i:integer;
        ymoy:real;

function MINIMUM(var a:tab;n:integer):real;

var i:integer;
    min:real;

begin
min:=a[1];
i:=1;
while i<n do
 begin
  i:=i+1;
  if a[i]<min then min:=a[i];
 end;
minimum:=min;
end;

(*************************************************************************************)

function MAXIMUM(var a:tab;n:integer):real;

var i:integer;
    max:real;

begin
max:=a[1];
i:=1;
while i<n do
 begin
  i:=i+1;
  if a[i]>max then max:=a[i];
 end;
maximum:=max;
end;

(*************************************************************************************)

 procedure ORIGIMP(var xmin,xmax,ymin,ymax,oax,oay:real);

     begin
        oax:=0;
        oay:=0;
        if xmin*xmax>0 then if xmin>0 then xmin:=0 else xmax:=0;
        if ymin*ymax>0 then if ymin>0 then ymin:=0 else ymax:=0;
     end;

(*************************************************************************************)
```

```
procedure ORIGPIMP(xmin,xmax,ymin,ymax:real;var oax,oay:real);

    begin
       oax:=0;
       oay:=0;
       if xmin*xmax>0 then if xmin<0 then oax:=xmax else oax:=xmin;
       if ymin*ymax>0 then if ymin<0 then oay:=ymax else oay:=ymin;
    end;

{*****************************************************************************}

procedure CAL1ABCD(xmin,xmax,ymin,ymax:real;var a,b,c,d:real);

    begin
     if (xmax-xmin) <>0 then a:=(amax-amin)/(xmax-xmin) else a:=0;
     b:=amin-a*xmin+xd;
     if (ymax-ymin) <>0 then c:=(bmax-bmin)/(ymax-ymin) else c:=0;
     d:=bmin-c*ymin+yd;
    end;

{*****************************************************************************}

procedure CAL2ABCD(xmin,xmax,ymin,ymax:real;var a,b,c,d:real);

var pgvx,pgvy:real;

    begin
    if abs(ymax) >= abs(ymin) then
     pgvy:=abs(ymax)
    else
     pgvy:=abs(ymin);
    if abs(xmax) >= abs(xmin) then
     pgvx:=abs(xmax)
    else
     pgvx:=abs(xmin);
    if (pgvy > pgvx) then
     begin
      if (ymax-ymin) <>0 then c:=(imax-imin)/(ymax-ymin) else c:=0;
      a:=c;
      b:=imin-a*xmin+xd;
      d:=imin-c*ymin+yd;
     end;
    if (pgvy < pgvx) then
     begin
      if (xmax-xmin) <>0 then a:=(imax-imin)/(xmax-xmin) else a:=0;
      c:=a;
      b:=imin-a*xmin+xd;
      d:=imin-c*ymin+yd;
     end;
    if (pgvy = pgvx) then
     begin
      if (ymax-ymin) <>0 then c:=(imax-imin)/(ymax-ymin) else c:=0;
      if (xmax-xmin) <>0 then a:=(imax-imin)/(xmax-xmin) else a:=0;
      b:=imin-a*xmin+xd;
      d:=imin-c*ymin+yd;
     end;
    end;

{*****************************************************************************}
```

```
procedure CALAXES(oax,oay,a,b,c,d:real;var coax,coay:integer);

    begin
     coax:=round(oax*a+b);
     coay:=round(oay*c+d);
    end;

{*****************************************************************************}

procedure DESAXES(coax,coay:integer);

    begin
    dessindroite(coax,coay-maxy,coax,coay+maxy);
    dessindroite(coax-maxx,coay,coax+maxx,coay);
    end;

{*****************************************************************************}

procedure CALCOOR(var x,y:tab;n:integer;a,b,c,d:real;var xc,yc:tabn);

var i:integer;

begin
for i:=1 to n do
 begin
  xc[i]:=round(x[i]*a+b);
  yc[i]:=round(y[i]*c+d);
 end;
end;

{*****************************************************************************}

procedure CALAFX(coax,coay,ax,ay:integer);

const lc=8;
      hc=10;
begin
if coax div 2 > amax div 2  then ax:=lc else
                    ax:=maxx-(lc*(length(vx)+2));
if coay div 2 > bmax div 2  then ay:=coay+hc else
                    ay:=coay-hc;
end;

{*****************************************************************************}

procedure CALAFY(coax,coay,ax,ay:integer);

const lc=8;
      hc=10;

begin
if coay div 2 > bmax div 2 then ay:=hc else
                    ay:=maxy-hc;
if coax div 2 > amax div 2 then ax:=coax-lc*((length(vx)+2)) else
                    ax:=coax+lc;
end;

{*****************************************************************************}


    begin (*DESSINGRAF*)
    xmin:=minimum(x,n);
```

```
      xmax:=maximum(x,n);
      ymin:=minimum(y,n);
      ymax:=maximum(y,n);
      if abs(ymin-ymax) < resol then
       begin
         ymoy:=(ymin+ymax)/2;
         ymin:=ymoy;ymax:=ymoy;
         for i:=1 to n do
          y[i]:=ymoy;
       end;
      case pc1 of
      '1':origimp(xmin,xmax,ymin,ymax,oax,oay);
      '2':origpimp(xmin,xmax,ymin,ymax,oax,oay);
      end;
      case pc2 of
      '1':call1abcd(xmin,xmax,ymin,ymax,a,b,c,d);
      '2':call2abcd(xmin,xmax,ymin,ymax,a,b,c,d);
      end;
      calaxes(oax,oay,a,b,c,d,coax,coay);
      initturtle;
      desaxes(coax,coay);
      calafx(coax,coay,ax,ay);
      afficher(ax,ay,vx);
      calafy(coax,coay,ax,ay);
      afficher(ax,ay,vy);
      calcoor(x,y,n,a,b,c,d,xc,yc);
      dpoints(xc,yc,n);
      end;

(*************************************************************************************************)

begin
end.
```

```
(*$S+,N+ *)

unit UTILIO;

interface

uses (*$U #5:VARGLOB.CODE *) varglob;

procedure LIRENTIER(var entier:integer);
procedure LIREEL(var reel:real);

(*****************************************************************************)

implementation


type typec=(tbol,teol,chiffre);

var tcc:typec;
    cc:char;
    inv,poscc,lglgn:integer;
    ligne:string;
    err:boolean;


function LIRE(var cc:char;var tcc:typec):boolean;

  function TYPECH(ch:char):typec;
  begin
  if (ord(ch) >=48) and (ord(ch) < 58) then typech:=chiffre;
  end;

(*****************************************************************************)

begin
if poscc>=lglgn then
 begin
 cc:='#';
 tcc:=teol;
 end
else
 begin
  poscc:=poscc+1;
  cc:=ligne[poscc];
  tcc:=typech(cc);
 end;
lire:=cc<>'#';
end;

(*****************************************************************************)

procedure ERREUR(num:integer);
begin
writeln('ERREUR ',num);
err:=true;
end;

(*****************************************************************************)
```

```
procedure RECONSTITUE(var ent:real;var nbchlu:integer);
begin
ent:=0;
nbchlu:=1;
repeat
ent:=ent*10+(ord(cc)-48);
if lire(cc,tcc) and (tcc = chiffre) then nbchlu:=nbchlu+1
until tcc <> chiffre;
end;

(*****************************************************************************)

procedure LIRENTIER;

var ent,nbent,lent:integer;
    reel:real;

begin
repeat
err:=false;
poscc:=0;cc:=' ';tcc:=tbol;
readln(ligne);
lent:=0;
inv:=1;
lglgn:=length(ligne);
if lire(cc,tcc) then
 begin
  if cc='-' then
   begin
    if lire(cc,tcc) then
     begin
      inv:=-1;
      lent:=1;
     end
   end;
  if cc='+' then
   begin
    if lire(cc,tcc) then
     begin
      inv:=1;
      lent:=1;
     end;
   end;
  if tcc=chiffre then
   begin
    reconstitue(reel,nbent);
    if reel <= maxint then ent:=trunc(reel)
    else erreur(4);
    lent:=lent+nbent;
    if lent < lglgn then erreur(3);
   end
  else erreur(2);
 end
else erreur(1);
if err then write('ENTRER UN ENTIER : ')
else entier:=ent*inv
until not err;
end;

(*****************************************************************************)
```

```
procedure LIREEL;

const maxex=25;

var li,lr,le,ls:integer;
    partexp:integer;
    pin,pre,pex:real;
    partreel,rel:real;

procedure PARTIEXPO;

begin
if lire(cc,tcc) then
 begin
  if tcc=chiffre then
   begin reconstitue(rel,le);
         if rel <= maxex then partexp:=trunc(rel)
         else erreur(10);
         pex:=pe(10,partexp);
         le:=le+1;
         if (li+lr+le+ls) < lglgn then erreur(3);
   end
  else
   begin
    if cc='-' then
     begin
      if lire(cc,tcc) then
       begin
        if tcc=chiffre then
         begin
          reconstitue(rel,le);
          if rel <= maxex then partexp:=trunc(rel)
          else erreur(10);
          pex:=pe(10,(-partexp));
          le:=le+2;
          if (li+lr+le+ls) < lglgn then erreur(3);
         end
        else erreur(6);
       end
      else erreur(7);
     end
    else erreur(8);
   end;
  end
else erreur(9);
end;

(*****************************************************************************************)

procedure PARTIEREEL;

begin
if lire(cc,tcc) then
 begin
  if tcc=chiffre then
   begin
    reconstitue(partreel,lr);
    pre:=partreel/pe(10,lr);
    lr:=lr+1;
    if cc='E' then partiexpo
    else if (li+lr+ls) < lglgn then erreur(3);
   end
```

```
  else erreur(4);
 end
else erreur(5);
end;

(*****************************************************************************)

procedure PARTIENTIER;

begin

reconstitue(pin,li);
if cc='.' then partiereel
else
 if cc='E' then partiexpo
 else
  if (li+ls) < lglgn then erreur(3);
end;

(*****************************************************************************)

begin
repeat
readln(ligne);
err:=false;
poscc:=0;cc:=' ';tcc:=tbol;
lglgn:=length(ligne);
li:=0;lr:=0;le:=0;
pin:=0;pre:=0;pex:=1;
ls:=0;inv:=1;
if lire(cc,tcc) then
 begin
  if cc='-' then
   begin
    if lire(cc,tcc) then
     begin
      inv:=-1;
      ls:=1;
     end
   end;
  if cc='+' then
   begin
    if lire(cc,tcc) then
     begin
      inv:=1;
      ls:=1;
     end
   end;
  if tcc=chiffre then partientier
  else
   begin
   if cc='.' then partiereel
   else
    begin
     if cc='E' then
      begin
       partiexpo;
       pin:=1;
      end
     else erreur(2);
    end;
   end;
```

```
 end
else erreur(1);
if err then write('ENTRER UN REEL : ')
else reel:=(pin+pre)*pex*inv
until not err;
end;

(*************************************************************************************************)

begin
end.
```

```
(*$S+,N+ *)
unit UTILREG;

interface

uses transcendend,
     (*$U #5:VARGLOB.CODE *) varglob;


procedure RL(var x,y:tab;n,chreg:integer;var a,b,c,r:real);

(*************************************************************************************
)

implementation

const presup=0.99;
      preinf=0.05;


procedure RL;

var i:integer;
    xm,ym,sx,sy,sxy,sx2,sy2,s2x,s2y,sxx,syy,cxy:real;

  procedure SUITERL;

  begin
  if abs(r) > preinf then
   begin
    case chreg of
      1:begin
          a:=cxy;
          b:=-s2x;
          c:=s2x*ym-cxy*xm;
        end;
      2:begin
          a:=-s2y;
          b:=cxy;
          c:=s2y*xm-cxy*ym;
        end;
      3:begin
          a:=2*cxy;
          b:=-(s2x-s2y+sqrt((sqr(s2x-s2y))+(4*sqr(cxy))));
          c:=((-b)*ym)-(a*xm);
        end;
    end;
   end
  else
   begin
    if abs(s2x) < (1-presup) then
     begin
      a:=1;
      b:=0;
      c:=-xm;
     end;
    if abs(s2y) < (1-presup) then
    begin
     a:=0;
     b:=-1;
     c:=ym;
    end;
```

```
    end;
end;

{*****************************************************************************}

begin{RL#}
sx:=0;
for i:=1 to n do sx:=sx+x[i];
sy:=0;
for i:=1 to n do sy:=sy+y[i];
sxy:=0;
for i:=1 to n do sxy:=sxy+(x[i]*y[i]);
sx2:=0;
for i:=1 to n do sx2:=sx2+pe(x[i],2);
sy2:=0;
for i:=1 to n do sy2:=sy2+pe(y[i],2);
cxy:=(sxy-((sx*sy)/n))/(n-1);
xm:=sx/n;
ym:=sy/n;
s2x:=(sx2-(pe(sx,2)/n))/(n-1);
s2y:=(sy2-(pe(sy,2)/n))/(n-1);
if s2x > 0 then sxx:=sqrt(s2x) else sxx:=0;
if s2y > 0 then syy:=sqrt(s2y) else syy:=0;
if (sxx*syy)<>0 then r:=cxy/(sxx*syy) else r:=0;
suiter1;
end;

{*****************************************************************************}

begin
end.
```

```
(*$S+,N+ *)

unit UTILECHEL;

interface

uses transcendend,
     (*$U #5:VARGLOB.CODE *) varglob,
     (*$U #5:UTILIO.CODE *) utilio;

procedure TRANSECHEL(var x,y:tab;n:integer;var vx,vy:tstring;var xc,yc:tab;
                     var vxc,vyc:tstring;var chx,chy,ex,ey:integer);
function REALI(var ch:integer;var t:tab;n:integer):boolean;
procedure TRANSPO(var v,vc:tstring;choix,e:integer);
procedure TRANSFO(choix,e:integer;var t:tab;n:integer;var t2:tab);

(*****************************************************************************)

implementation

procedure TRANSFO;

procedure LOGA;

var i:integer;

begin
for i:=1 to n do t2[i]:=log(t[i]);
end;

(*****************************************************************************)

procedure LNP;

var i:integer;

begin
for i:=1 to n do t2[i]:=ln(t[i]);
end;

(*****************************************************************************)

procedure EXPO;

var i:integer;

begin
for i:=1 to n do t2[i]:=exp(t[i]);
end;

(*****************************************************************************)

procedure ARTG;

var i:integer;

begin
for i:=1 to n do t2[i]:=atan(t[i]);
end;

(*****************************************************************************)
```

```
procedure XEY;

var i:integer;

begin
for i:=1 to n do t2[i]:=pe(t[i],e);
end;

(*************************************************************************************)

procedure SINU;

var i:integer;

begin
for i:=1 to n do t2[i]:=sin(t[i]);
end;

(*************************************************************************************)

procedure COSI;

var i:integer;

begin
for i:=1 to n do t2[i]:=cos(t[i]);
end;

(*************************************************************************************)

procedure RACI;

var i:integer;

begin
for i:=1 to n do t2[i]:=sqrt(t[i]);
end;

(*************************************************************************************)

procedure DIXX;

var i:integer;

begin
for i:=1 to n do t2[i]:=exp(t[i]*ln(10));
end;

(*************************************************************************************)


begin
(*$R TRANSCENDEND *)
case choix of
2:sinu;
3:cosi;
4:raci;
5:loga;
6:xey;
7:lnp;
8:artg;
9:expo;
```

```
10:dixx;
end;
end;

(*****************************************************************************)

procedure TRANSFO;

var se:tstring;

begin
case choix of
1:vc:=v;
2:vc:=concat('SIN(',v,')');
3:vc:=concat('COS(',v,')');
4:vc:=concat(v,'^1/2');
5:vc:=concat('LOG(',v,')');
6:begin str(e,se);vc:=concat(v,'^',se);end;
7:vc:=concat('LN(',v,')');
8:vc:=concat('ARTAN(',v,')');
9:vc:=concat('EXP(',v,')');
10:vc:=concat('10^(',v,')');
end;
end;

(*****************************************************************************)

function REALI;

var err:boolean;
    i:integer;

begin
case ch of
1,2,3,6,8,9,10:reali:=true;
5,7:begin err:=false;
          reali:=true;
          i:=0;
          while (i<n) and (not err) do
           begin
            i:=i+1;
            if t[i] <= 0 then err:=true;
           end;
          if err then
           begin
        if ch=5 then writeln('LOG D''UN NOMBRE NEGATIF OU NUL');
        if ch=7 then writeln('LN D''UN NOMBRE NEGATIF OU NUL');
           reali:=false;
           end;
    end;
4:begin err:=false;
        reali:=true;
        i:=0;
        while (i<n) and (not err) do
         begin
          i:=i+1;
          if t[i] < 0 then err:=true;
         end;
        if err then
         begin
         writeln('RACINE CARREE D''UN NOMBRE NEGATIF');
```

```
          reali:=false;
          end;
   end;
end;
end;

{*****************************************************************************}

procedure TRANSECHEL;

const maxexpo=25;

procedure MENUECHEL(c:char);

var ok:boolean;

begin
writeln('[1]:',c);
writeln('[2]:SIN(',c,')');
writeln('[3]:COS(',c,')');
writeln('[4]:1/',c);
writeln('[5]:LOG(',c,')');
writeln('[6]:',c,'^E');
writeln('[7]:LN(',c,')');
writeln('[8]:ARTG(',c,')');
writeln('[9]:EXP(',c,')');
writeln('[10]:10^',c);
case c of
'X':begin ok:=false;
          repeat
           readln(chx);
           if reali (chx,x,n) then
            begin
             if chx=6 then
              begin
               repeat
                write('EXPOSANT : ');
                lirentier(ex)
               until (ex < maxexpo);
              end;
             transfo(chx,ex,x,n,xc);
             transpo(vx,vxc,chx,ex);
             ok:=true;
            end
          until ok;
    end;
'Y':begin ok:=false;
          repeat
           readln(chy);
           if reali (chy,y,n) then
            begin
             if chy=6 then
              begin
               repeat
                write('EXPOSANT : ');
                lirentier(ey)
               until (ey < maxexpo);
              end;
             transfo(chy,ey,y,n,yc);
             transpo(vy,vyc,chy,ey);
             ok:=true;
```

```
            end
          until ok;
    end;
end;
end;

(*****************************************************************************)

begin
menuechel('X');
menuechel('Y');
end;

(*****************************************************************************)

begin
end.
```

```
(*$S+ *)
program COORD1;

uses transcendend,turtlegraphics,
     (*$U #5:VARGLOB.CODE *) varglob,
     (*$U #5:ANALCHIMIE.CODE *) analchimie,
     (*$U #5:VALSTA.CODE *) valsta,
     (*$U #5:UTILIO.CODE *) utilio,
     (*$U #5:CALCHIM1.CODE *) calchim1,
     (*$U #5:FONCINF.CODE *) foncinf,
     (*$U #5:UTILDES.CODE *) utildes,
     (*$U #5:UTILREG.CODE *) utilreg,
     (*$U #5:UTILECHEL.CODE *) utilechel,
     (*$U #5:INFERER.CODE *) inferer;

var ch:char;
    temp,k:real;
    tco:tabf;
    tcs,tch:tabn;
    nc,nr:integer;
    tyre:typere;
    i,np:integer;
    tequ:tabeq;

begin
(*$N+ *)
(*$R VARGLOB.TURTLEGRAPHICS *)
page(output);
writeln('INTRODUIRE UNE EQUATION');
while not equachim(nc,nr,tco,tcs,tch,tyre) do
 begin
  readln;
  page(output);
  writeln('INTRODUIRE UNE EQUATION');
 end;
page(output);
write('TEMPERATURE : ');
lireel(temp);
write('NP : ');
lirentier(np);
if getke(tco,tcs,nr,nc,temp,k) then
 begin
  ch:='1';
  pinfer(nc,np,tco,ch,tequ);
  for i:=1 to pe(2,(nc-1)-1) faire
   writeln('EQUATION',i,' : ',tequ[i]);
 end;
end.
```

```
(*$S+ *)
program COORD2;

uses transcendend,turtlegraphics,
     (*$U #5:VARGLOB.CODE *) varglob,
     (*$U #5:ANALCHIMIE.CODE *) analchimie,
     (*$U #5:VALSTA.CODE *) valsta,
     (*$U #5:UTILIO.CODE *) utilio,
     (*$U #5:CALCHIM1.CODE *) calchim1,
     (*$U #5:FONCINF.CODE *) foncinf,
     (*$U #5:UTILDES.CODE *) utildes,
     (*$U #5:UTILREG.CODE *) utilreg,
     (*$U #5:UTILECHEL.CODE *) utilechel,
     (*$U #5:INFERER.CODE *) inferer;

var ch:char;
    i,np,nc,nr,nid:integer;
    tco,tid:tabf;
    tcs,tch:tabn;
    tyre:typere;
    dh0,ds0:real;
    tequ:tabeq;

begin
(*$N+ *)
(*$R VARGLOB,TURTLEGRAPHICS *)
page(output);
writeln('INTRODUIRE UNE EQUATION');
while not equachim(nc,nr,tco,tcs,tch,tyre) do
 begin
  readln;
  page(output);
  writeln('INTRODUIRE UNE EQUATION');
 end;
page(output);
write('NP :');
lirentier(np);
if getdho(tco,tcs,nr,nc,dh0) and getds0(tco,tcs,nr,nc,ds0) then
 begin
  tid[1]:='T';tid[2]:='K';nid:=2;
  pinfer(nid,n,tid,ch,tequ);
 end;
end.
```

```
(*$S+ *)
program COORD3;

uses transcendend,turtlegraphics,
   (*$U #5:VARGLOB.CODE *) varglob,
   (*$U #5:ANALCHIMIE.CODE *) analchimie,
   (*$U #5:VALSTA.CODE *) valsta,
   (*$U #5:CALCHIM1.CODE *) calchim1,
   (*$U #5:UTILIO.CODE *) utilio,
   (*$U #5:CALCHIM2.CODE *) calchim2,
   (*$U #5:UTILDES.CODE *) utildes;

var fini:boolean;
    rep:char;
    dg0,tmin,tmax,tstep,dh0,ds0:real;
    i,j,np,nc,nr:integer;
    tch,tcs:tabn;
    tco:tabf;
    tyre:typere;
    ci,ce,tt,tk:tab;
    ct:array[1..lmtab2,1..lmtab5] of real;
    vx,vy:tstring;


procedure VARDOM(id:string;np:integer;var vmin,vmax,vstep:real);

begin
write('UNE VALEUR INITIALE POUR ',id,' : ');
lireel(vmin);
write('UNE VALEUR FINALE POUR ',id,' : ');
lireel(vmax);
vstep:=(vmax-vmin)/(np-1);
end;

(*****************************************************************************)


procedure KFT(n:integer;tmin,tstep,dh0,ds0:real;var tk,tt:tab);

begin
for i:=1 to np do
      begin
       tt[i]:=tmin+((i-1)*tstep);
       dg0:=valdg0(dh0,ds0,tt[i]);
       tk[i]:=3*tt[i];
      end;
end;

(*****************************************************************************)


procedure CONCFT;

var fini1,fini2:boolean;
    rep1,rep2,pc1,pc2:char;
    cc:tab;
    ny:integer;
    no:tab5;
    resol:real;

begin
fini1:=false;
```

```
repeat
fini:=false;
repeat
write('Y : ');
readln(vy);
if exveta(vy,tco,nc,ny) then fini:=true
until fini;
for i:=1 to np do  cc[i]:=ct[i,ny];
resol:=0.000000001;
fini2:=false;
repeat
writeln('[1]:TENIR COMPTE DE L''ORIGINE');
writeln('[2]:NE PAS TENIR COMPTE DE L''ORIGINE');
repeat read(keyboard,pc1) until (pc1 in ['1','2']);
writeln(pc1);
writeln('[1]:PAS DIFFERENTS');
writeln('[2]:PAS IDENTIQUES');
repeat read(keyboard,pc2) until (pc2 in['1','2']);
writeln(pc2);
dessingraf(tt,cc,np,vx,vy,pc1,pc2,resol);
readln;
textmode;
write('UN AUTRE CHOIX DE GRAPHIQUE : ');
readln(rep2);
if rep2 = 'N' then fini2:=true
until fini2;
write('UN AUTRE COMPOSE : ');
readln(rep1);
if rep1 = 'N' then fini1:=true
until fini1;
end;


begin
(*$N+ *)
(*$R TURTLEGRAPHICS,TRANSCENDEND,VARGLOB *)
fin:=false;
repeat
page(output);
writeln('INTRODUIRE UNE EQUATION');
while not equachim(nc,nr,tco,tcs,tch,tyre) do
 begin
  readln;
  page(output);
  writeln('INTRODUIRE UNE EQUATION');
 end;
if getdh0(tco,tcs,nr,nc,dh0) and getds0(tco,tcs,nr,nc,ds0) then
   begin
    writeln;
    writeln('DH0 : ',dh0);
    writeln('DS0 : ',ds0);
    writeln;
    write('NOMBRE DE POINTS : ');
    lirentier(np);
    while np > lmtab do
     begin
      write('NOMBRE DE POINTS : ');
      lirentier(np);
     end;
    writeln;
    vx:='T';
    vardom(vx,np,tmin,tmax,tstep);
```

```
      writeln;
      kft(np,tmin.tstep,dh0,ds0,tk,tt);
      getconcini(tco,nr,ci);
      for i:=1 to np do
       begin
        getconcequil(tcs,ci,nr,nc,tk[i],ce);
        for j:=1 to nc do ct[i,j]:=ce[j];
       end;
      concft;
     end;
write('ON CONTINUE : ');
readln(rep);
if rep='N' then fin:=true
until fin;
end.
```

```
(*$S+*)
program ACCESDON;

uses (*$U #5:VARGLOB.CODE *) varglob,
     (*$U #5:VALSTA.CODE *) valsta,
     (*$U #5:ANALCHIMIE.CODE *) analchimie,
     (*$U #5:UTILIO.CODE *) utilio;

var a:article;
    i:identifiant;
    at:attribut;
    choix,rep,reponse:char;
    fin,fini:boolean;
    compenre:integer;

procedure MENU2;

begin
writeln;
writeln('[1]:ENREGISTRER');
writeln('[2]:CONSULTER SEQUENTIELLEMENT');
writeln('[3]:CONSULTER UN COMPOSE');
writeln('[4]:MODIFIER UN COMPOSE');
writeln('[5]:SUPPRIMER UN COMPOSE');
writeln('[6]:PURGER LE FICHIER');
writeln('[7]:TRIER LE FICHIER');
writeln('[8]:FIN');
writeln;
end;

procedure AJOUTER(sf:string);

begin page(output);
      writeln('COMPOSE : ');
      while (not compchim(a.id)) do
       begin
        readln;
        page(output);
        writeln('COMPOSE : ');
       end;write('NOF : ');
      lireel(a.att.nof);
      write('SOF : ');
      lireel(a.att.sof);
      writeln;
      if not exisms(f,f1,a.id,a.att) and not exisms(f,f2,a.id,a.att) then
       enregis(f,sf,a)
      else writeln('CE COMPOSE EST DEJA ENREGISTRE');
      writeln;
end;

procedure CONSULSEQ(sf:string);

begin
consfseq(f,sf);
writeln;
end;
```

```
procedure CONSULID(sf:string);

begin page(output);
      writeln('COMPOSE A CONSULTER : ');
      while (not compchim(i)) do
       begin
        readln;
        page(output);
        writeln('COMPOSE A CONSULTER : ');
       end;
      writeln;
      consid(f,sf,i);
      writeln;
end;

procedure CHANGER(sf:string);

begin page(output);
      writeln('COMPOSE A MODIFIER : ');
      while (not compchim(i)) do
       begin
        readln;
        page(output);
        writeln('COMPOSE A MODIFIER : ');
       end;
      writeln;
      if exisms(f,sf,i,at) then
       begin
        a.id:=i;
        write('HOF : ');
        lireel(a.att.hof);
        write('SOF : ');
        lireel(a.att.sof);
        modific(f,sf,a);
       end
      else writeln('LE COMPOSE A MODIFIER N''EXISTE PAS ');
      writeln;
end;

procedure DETRUIRE(sf:string);

begin page(output);
      writeln('COMPOSE A SUPPRIMER : ');
      while (not compchim(i)) do
       begin
        readln;
        writeln('COMPOSE A SUPPRIMER : ');
        page(output);
       end;
      writeln;
       if exisms(f,sf,i,at) then
       suppres(f,sf,i)
      else writeln('LE COMPOSE A SUPPRIMER N''EXISTE PAS ');
      writeln;
end;
```

```
procedure NETTOYAGE(sf:string);

begin write('ETES VOUS SUR DE VOTRE OPERATION : ');
      readln(reponse);
      if reponse = 'O' then
       begin rewrite(f,sf);
             close(f,lock);
       end;
      writeln;
end;


procedure MENU1;

begin
writeln;
writeln('[1]:MANIPULATION SUR LE FICHIER CONTENANT LES');
writeln('    COMPOSES MEMORISES EN MEMOIRE SECONDAIRE');
writeln('[2]:MANIPULATION SUR LE FICHIER CONTENANT LES');
writeln('    COMPOSES MEMORISES EN MEMOIRE CENTRALE');
writeln('[3]:FIN');
writeln;
end;


begin
fini:=false;
repeat
menu1;
readln(choix);
case choix of
'1':begin fin:=false;
          repeat menu2;
                 readln(reponse);
                 page(output);
                 case reponse of
                 '1':begin
                      rep:='O';
                      while rep <> 'N' do
                       begin ajouter(f1);
                             write('ON CONTINUE : ');
                             readln(rep);
                       end;
                     end;
                 '2':consulseq(f1);
                 '3':consulid(f1);
                 '4':changer(f1);
                 '5':detruire(f1);
                 '6':nettoyage(f1);
                 '7':begin
                      compenre:=compter(f,f1);
                      if (compenre > 1) then  tri(f,f1,compenre);
                     end;
                 '8':fin:=true;
                 end
          until fin
    end;
'2':begin fin:=false;
      repeat menu2;
             readln(reponse);
             page(output);
             case reponse of
```

```
                '1':begin
                      compenre:=compter(f,f3);
                      while compenre < lmtabcomp do
                       begin ajouter(f3);
                             compenre:=compenre+1;
                             write('ON CONTINUE ;');
                             readln(rep);
                             if rep='N' then compenre:=lmtabcomp
                       end;
                      end;
                '2':consulseq(f3);
                '3':consulid(f3);
                '4':changer(f3);
                '5':detruire(f3);
                '6':nettoyage(f3);
                '7':begin
                      compenre:=compter(f,f3);
                      if compenre > 1 then tri(f,f3,compenre);
                     end;
                '8':fin:=true;
                end
      until fin
    end;
 '3':fini:=true;
end
until fini;
end.
```